AEDC-TR-10-T-36

PA No. AEDC2011-219

2D Mesh Manipulation

James S. Masters
Aerospace Testing Alliance

November 2011

Final Report for Period October 2009 – January 2011

ARNOLD ENGINEERING DEVELOPMENT CENTER
ARNOLD AIR FORCE BASE, TENNESSEE
AIR FORCE MATERIEL COMMAND
UNITED STATES AIR FORCE

NOTICES
When US Government drawings, specifications, or other data are used for any purpose other than a definitely related Government procurement operation, the Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the Government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.
Qualified users may obtain copies of this report from the Defense Technical Information Center.
References to named commercial products in this report are not to be considered in any sense as an endorsement of the product by the United States Air Force or the Government.

DESTRUCTION NOTICE

For unclassified, limited documents, destroy by any method that will prevent disclosure or reconstruction of the document.

APPROVAL STATEMENT

Prepared by:

JAMES S. MASTERS
Aerospace Testing Alliance

Reviewed by:

TERRANCE M. DUBREUS, Ph.D.
AEDC/TTSY

Approved by:

SCOTT W. WALTERMIRE
Technical Director
Test Technology

REPORT DOCUMENTATION PAGE	Form Approved OMB No. 0704-0188

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED *(From – To)*
xx-11-2011	Final Report	October 2009 – January 2011

4. TITLE AND SUBTITLE
2D Mesh Manipulation

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Masters, James S.
Aerospace Testing Alliance

5d. PROJECT NUMBER
11808

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Aerospace Testing Alliance (ATA)

8. PERFORMING ORGANIZATION REPORT NO.
AEDC-TR-10-T-36

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Statement A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
Available in the Defense Technical Information Center (DTIC).

14. ABSTRACT

Unstructured methods for region discretization have become common in computational fluid dynamics (CFD) analysis because of certain benefits they have over structured meshes (where the discretization of the domain reflects some type of consistent geometrical regularity). Unstructured meshes are powerful tools for efficiently defining complex geometries and also lend themselves to analysis techniques such as adaptive mesh refinement (AMR) where the properties of the mesh adapt to the properties of the flow that is being analyzed. They also are useful for adapting a mesh to a geometry that is being optimized to exhibit desired characteristics. As unstructured methods have grown in popularity, mesh manipulation techniques that have traditionally been reserved for structured meshes have started migrating to the world of unstructured meshing as well. One such technique is the application of Winslow elliptic smoothing equations to unstructured meshes. It has been shown that it is not necessary for the computational space of the entire mesh to be constructed as an overarching system; rather, each node in computational space can be treated as an individual virtual control volume. This allows Winslow equations to be utilized even if the original physical mesh is of low quality. Traditional Winslow equations have been shown to be ideal for smoothing nonboundary nodes in inviscid regions but are of limited use in other situations. Modifications to the implementation of the computational space allow Winslow equations to be extended so that they can also be applied to boundary nodes and to highly anisotropic viscous regions of unstructured meshes.

15. Subject Terms
Unstructured mesh, elliptic smoothing, adaptive mesh refinement

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19A. NAME OF RESPONSIBLE PERSON
A. REPORT	B. ABSTRACT	C. THIS PAGE			James S. Masters
Unclassified	Unclassified	Unclassified	Same as Report	93	19B. TELEPHONE NUMBER *(Include area code)* (931) 454-7739

This page is intentionally left blank.

CONTENTS

FIGURES

Figure

APPENDICES

1.0 INTRODUCTION

In any engineering field it is important to have the ability to break up a given system into manageable parts in order to explore and investigate the interesting and dynamic aspects of the system. This is true for the study of thermodynamic systems or for the use of finite-element analysis to study static and dynamic properties of parts and facilities. It is also true within the field of fluid mechanics and especially the subdiscipline of computational fluid dynamics (CFD).

The analysis of aerodynamic systems is often broken down into three separate but complimentary categories: flight-test, ground-test (wind tunnel testing), and CFD. CFD deals directly with applying the laws of conservation (mass, momentum, and energy) to an aerodynamic system. It relies on a logical and effective breakup of the system of interest in order to properly apply the conservative laws of fluid dynamics to the system in a way that properly captures the necessary physics to realistically define the system.

Mesh generation is the discipline that explores and analyzes the breakup (known as discretization) of a system into finite discrete segments so that the characteristics of the overarching system can be explored using the Navier-Stokes equations (or, in the case of inviscid flow, the Euler equations). This system of equations – named after the Frenchman Claude-Louis Navier and the Englishman Gabriel Stokes – are a system of partial differential equations. Analytical solutions to partial differential equations involve closed-form expressions which give the variation of the dependent variables continuously throughout a domain (Ref.1)[1]. However, to analyze a physical system computationally, both the defining equations and the domain must be discretized. Discretization of the equations is accomplished by replacing the original system of partial differential equations with a system of algebraic equations that can be solved to calculate the values of the flowfield variables (pressure, temperature, velocity, etc.) at discrete points (often referred to as grid points, mesh points, or nodes). Depending on the way the equations are discretized, a finite difference or finite volume numerical solution is obtained. In contrast to differential equations, numerical solutions generally give answers only at discrete points in the domain; effectively generating these discrete grid point locations is at the heart of mesh generation and manipulation.

There are many ways that a region where the flow is to be analyzed might be discretized. Traditionally, the region has been discretized using structured meshes, where the discretization of the domain reflects some type of consistent geometrical regularity. However, because of their flexibility with capturing real-world geometry, unstructured meshes (where mesh points are placed in the flowfield in a very irregular fashion) are being utilized more frequently (Ref. 1). As computational power and storage increase and as unstructured meshes become better understood and handled with increasing skill, mesh manipulation techniques that have in the past been reserved for structured meshes are migrating to the world of unstructured meshing as well. One such technique is the application of Winslow elliptic smoothing equations to a mesh.

1.1 MESH SMOOTHING AND WINSLOW EQUATIONS

The Winslow elliptic smoothing equations, first proposed by Alan Winslow in 1967 (Ref. 2), are derived from Poisson's Equation (or Laplace's Equation for the homogeneous case) for a parameter distribution over a region. In structured meshing there is an implicit computational space that lends itself well to the application of the Winslow equations. However, no such implicit computational space exists for unstructured meshes, and the computational space must, therefore, be explicitly defined. It can be shown that a computational space can be constructed using the initial discretized physical space. This works well for many applications and often

provides a well-defined connectivity and spatial definition for a mesh that is applicable even if the geometry that defined the mesh was then allowed to move. However, this also creates the requirement that a valid initial mesh exists. This is not always the case; an initial unstructured mesh might not exist and, if one does exist, it is possible that there might exist sections in the mesh that are invalid due to mesh crossing, negative volumes, or other problems. Using the original mesh as the computational space is also problematic if there is a requirement that certain mesh characteristics (such as viscous spacing) need to be varied.

It is apparent that there are significant limitations with using the original mesh as the computational space, but a mathematically expedient way of defining a general overarching unstructured computational space has proven elusive. Because the unstructured mesh connectivity needs to be explicitly defined and the valence count (the number of connected nodes or neighbors) varies throughout the domain, no theory currently exists to generate a global computational space that reflects a consistent geometric regularity as was done with structured meshes. Many difficulties that were due to the lack of a global unstructured computational space were alleviated when it was shown that it was not necessary for the entire computational mesh to be constructed as an overarching system of nodes and elements and that each node in computational space could be isolated from the overarching system as a virtual control volume and coupled only though the coordinates in physical space, which are treated as free variables (Refs. 3 and 4). These virtual control volumes can be constructed with ideal shapes and uniform quality.

A physical region is discretized using structured meshes by breaking up the region into quadrilaterals (quads) in two dimensions and hexahedra (hexes) in three dimensions. The simplest way to discretize a physical region using unstructured meshes is to break up the region into triangles in two dimensions and tetrahedra (tets) in three dimensions. There are many other ways to discretize a region using unstructured meshes, but this document will only address the simplex geometry elements. Examples of a simple two-dimensional region that has been discretized using both structured and unstructured methodologies, as well as the corresponding computational spaces, are shown in Figs. 1 and 2. The convention used within this document is that physical space is generally displayed with a red mesh while computational space is generally displayed with a black or gray mesh; this convention is illustrated in Figs. 1 and 2. Additional information concerning grid metrics, which describe the relationship (or mapping) between physical and computational space, is assembled in Appendix A.

When applied to an unstructured mesh, the Winslow elliptic smoothing equations allow mesh points to be moved and smoothed to conform to a moving (or nonmoving) surface. The Winslow equations deal with derivatives of physical space (x,y) with respect to computational space (ξ,η). A detailed derivation of the Winslow equations is demonstrated in Appendix B, and it is interesting to note how much more complex the equations become when they are transformed from the relatively simple form with respect to coordinates in physical space compared to the form where the derivatives are with respect to the computational coordinates.

In Fig. 2 the nodes, which correspond to the four nodes shown as black dots in physical space, are broken off from the overarching system and are coupled only through the coordinates in physical space. Each node will have a unique computational space similar to the ones shown in the figure. Note that the computational space for a boundary node does not have a complete neighbor-node stencil.

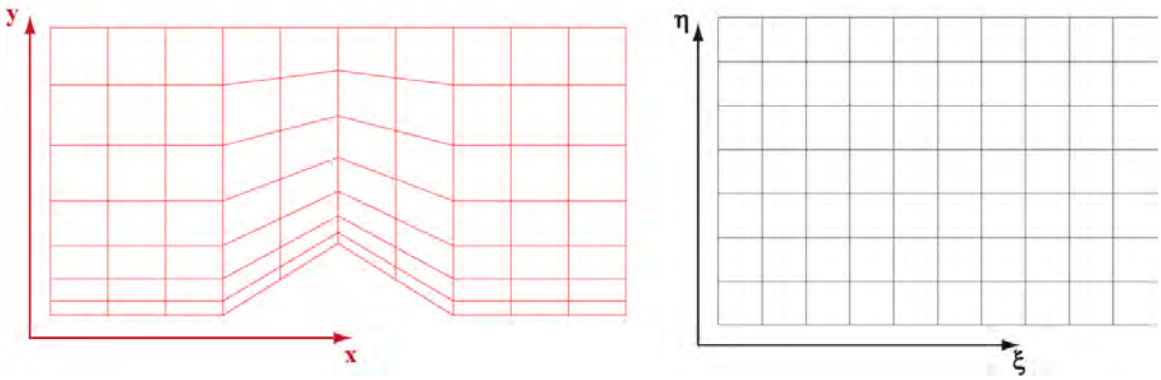

Figure 1. Structured Mesh in Physical and Computational Space

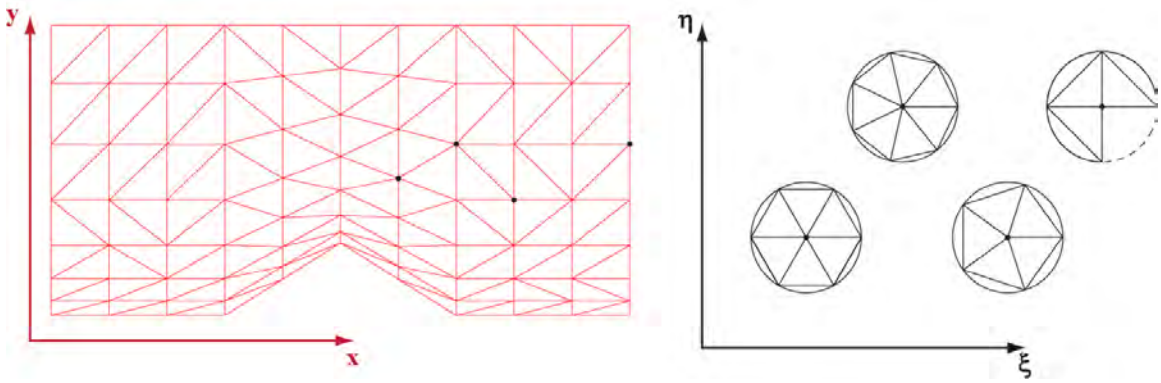

Figure 2. Unstructured Mesh in Physical and Computational Space

1.2 EXTENSION OF WINSLOW EQUATIONS

For an unstructured mesh, the central node in each of the individual virtual control volumes in computational space is surrounded by neighboring nodes using equal angles and equal edge-lengths, and it is necessary that the central node be fully surrounded by neighboring nodes for the computational stencil to be complete. (A note on terminology: the node-of-interest – i.e., the node being smoothed at a given instant – will often be referred to in this document as the central node and is always located at the origin in computational space.) The characteristics of the Winslow equations implemented using this method make them ideal for smoothing inviscid mesh elements in the interior region of a mesh because the equations drive the mesh elements to display an isotropic behavior; that is, the triangular mesh elements in physical space will all become as close to equilateral as possible within the constraints of the overall system.

The need for a complete computational neighbor-node stencil and the tendency to drive mesh elements to exhibit isotropic behavior, however, makes the conventional Winslow methodology unsuitable to two situations that are common in fluid mechanics (Ref. 5). The first situation is where the nodes on the boundaries need to move to accommodate the movement of the interior nodes. Often, keeping the boundary nodes static while interior nodes move will result in mesh elements that exhibit a sufficient amount of skew to cause the numerical solutions generated on the mesh to become unstable or unreliable. The second situation is one where the viscous properties of the flow are important. In the viscous boundary-layer region, the flowfield variables are changing very rapidly normal to the surface, but much more mildly in the direction parallel to the surface. In order to capture the viscous boundary layer in an efficient manner, it is

necessary that the mesh elements near the viscous surface have high aspect ratios. Otherwise, the resolution required to capture the gradients in the normal direction will result in a potentially prohibitive number of grid points.

These two common situations are addressed in detail in this report in Sections 6 and 7. A methodology is presented and described that will make it possible to apply the Winslow elliptic smoothing equations to a node on a boundary. Implementation of this methodology allows the boundary nodes to float and greatly improve the mesh in certain situations. Several examples are shown that illustrate the benefit of implementing the Winslow equations such that the surface nodes are allowed to float. A methodology is also presented that allows the Winslow equations to be applied to viscous regions of a mesh, where it is required that the mesh be highly anisotropic. Examples of this implementation are illustrated as well.

1.3 MESH REFINEMENT AND OPTIMIZATION

Two areas where the Winslow equations can be used in conjunction with other analysis techniques are adaptive mesh refinement (AMR) and optimization. AMR is employed with the purpose of reducing the computational cost of generating a flow solution while maintaining a given level of accuracy for the solution. In a finite-element context, adaptive mesh refinement is generally classified into three categories: h-refinement (enrichment), r-refinement (movement), and p-refinement (reconnection).

H-refinement schemes often use an error estimator to determine regions of the grid where the solution is underresolved and add elements locally to improve the resolution. This is done with the intent of improving the accuracy of the solution. While h-refinement schemes have demonstrated the ability to improve solution accuracy through local refinement, there is added overhead due to the increased number of mesh elements (Ref. 6). The p-refinement approach increases the element order by increasing the degree of the piecewise polynomials over the triangular mesh elements. The r-refinement approach involves the redistribution of points. R-refinement involves moving the nodes of the mesh while maintaining the cell connectivity and the size of the mesh (Ref. 7).

R-refinement is the category that includes mesh smoothing. Conceptually, r-refinement is relatively straightforward, but it can produce highly skewed cells and prohibitively small (or negative) cell volumes if not performed correctly. AMR generally refers to adapting a mesh to a particular flow solution but herein deals also with adapting a mesh to changes in geometry.

Design optimization may be performed to improve the performance of a given design operating under a certain set of conditions. An efficient way to perform a design optimization on a 2D mesh is to couple an optimization code with a 2D CFD solver and a mesh movement/refinement code. The concept of optimization is based on finding a minimum of some function. A design reaching an optimum state implies that the function for which the optimization has been performed has been minimized and the derivates of the function are zero. By employing sensitivity derivatives based on the impact of a given set of design variables, a design can be optimized to better perform a particular mission. A mesh manipulation method may be used to manipulate the geometry to minimize the function or modify the volume grid to improve the accuracy of the solution.

2.0 BACKGROUND

The gradient in an unstructured element is derived by manipulating the Divergence Theorem (often alternately referred to as the Gauss Divergence Theorem or, in 2D, as Green's Theorem). The Divergence Theorem states that if there is a solid region (i.e., a control volume) whose boundary surface has positive orientation (outward facing surface normals, $\vec{n}$) and $\vec{F}$ is a vector field whose component functions have continuous partial derivatives on an open region containing the control volume (Ref. 8), then:

$$\iiint \nabla \cdot \vec{F}\, dV = \iint \vec{F} \cdot \vec{n} dS \tag{2.1}$$

2.1 GRADIENTS IN AN UNSTRUCTURED MESH

Equation (2.1) is the Divergence Theorem for a three-dimensional (3D) control volume. In two dimensions, the Divergence Theorem can be simplified to Eq. (2.2).

$$\iint \nabla \cdot \vec{F}\, dA = \int \vec{F} \cdot \vec{n} dS \tag{2.2}$$

The control volume in 2D is effectively a "control area" but will continue to be referred to as a control volume herein. Consider a 2D domain broken up into an unstructured mesh comprised of triangles similar to the one shown below in Fig. 3. Each triangle becomes a control volume, and the gradient within the control volume (which is treated as constant throughout the control volume and discontinuous at the surface of the control volume) can be found by manipulating the Divergence Theorem. Note that because the gradient is constant within the control volume, the mesh must be refined into smaller control volumes if a more precise gradient is desired in a given region.

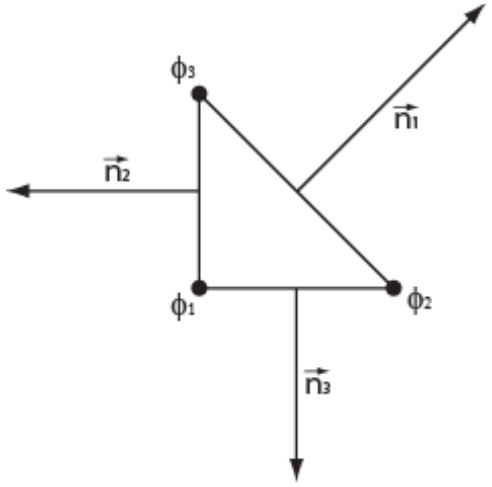

Figure 3. Two-Dimensional Triangular Control Volume with Outward Facing Normals

A generic scalar function (denoted as ϕ) is used to illustrate the process for calculating the gradient in a triangular control volume. Any scalar function that is continuous in the region can be calculated using the following methodology. The gradient of a scalar function is a vector function. In two dimensions, this gradient can be written as:

$$\nabla\phi = \begin{bmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{bmatrix}$$

The average values of the components of the gradient in a 2D control volume can be calculated using Eq. (2.3).

$$\begin{aligned} \overline{\frac{\partial\phi}{\partial x}} &= \frac{1}{A}\iint \frac{\partial\phi}{\partial x}\,dA \\ \overline{\frac{\partial\phi}{\partial y}} &= \frac{1}{A}\iint \frac{\partial\phi}{\partial y}\,dA \end{aligned} \tag{2.3}$$

The divergence of a vector, $\nabla\cdot\vec{F}$, is a scalar and can be related to either of the two components of the 2D gradient as follows:

$$\vec{F} = \begin{bmatrix} \phi \\ 0 \end{bmatrix} \Rightarrow \nabla\cdot\vec{F} = \frac{\partial\phi}{\partial x} + 0 = \frac{\partial\phi}{\partial x}$$

$$\vec{F} = \begin{bmatrix} 0 \\ \phi \end{bmatrix} \Rightarrow \nabla\cdot\vec{F} = 0 + \frac{\partial\phi}{\partial y} = \frac{\partial\phi}{\partial y}$$

$$\iint \frac{\partial\phi}{\partial x}\,dA = \underbrace{\iint \nabla\cdot\vec{F}\,dA = \oint \vec{F}\cdot\vec{n}\,dS}_{\text{Divergence Thm}} = \int \begin{bmatrix} \phi \\ 0 \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \end{bmatrix} dS = \int \phi\, n_x\, dS$$

$$\iint \frac{\partial\phi}{\partial y}\,dA = \underbrace{\iint \nabla\cdot\vec{F}\,dA = \oint \vec{F}\cdot\vec{n}\,dS}_{\text{Divergence Thm}} = \int \begin{bmatrix} 0 \\ \phi \end{bmatrix} \cdot \begin{bmatrix} n_x \\ n_y \end{bmatrix} dS = \int \phi\, n_y\, dS$$

$$\begin{aligned} \overline{\frac{\partial\phi}{\partial x}} &= \frac{1}{A}\iint \frac{\partial\phi}{\partial x}\,dA = \frac{1}{A}\int \phi\, n_x\, dS \\ \overline{\frac{\partial\phi}{\partial y}} &= \frac{1}{A}\iint \frac{\partial\phi}{\partial y}\,dA = \frac{1}{A}\int \phi\, n_y\, dS \end{aligned}$$

Dropping the overbar for simplicity, with the understanding that the gradient is treated as constant within the control volume, the equation for the gradient becomes:

$$\nabla\phi = \begin{bmatrix} \dfrac{\partial\phi}{\partial x} \\ \dfrac{\partial\phi}{\partial y} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{A}\int \phi\, n_x\, dS \\ \dfrac{1}{A}\int \phi\, n_y\, dS \end{bmatrix} \tag{2.4}$$

For a triangular area, the surface integrals shown on Eq. (2.4) can be discretized by summing the values of the integrand on each of the triangle's three edges. Using the x-component for example, the gradient is discretized in Eq. (2.5).

$$\frac{1}{A}\int \phi\, n_x\, dS \approx \frac{1}{A}\sum_{i=1}^{3} \bar{\phi}_i\, \hat{n}_{xi}\, \Gamma_i \tag{2.5}$$

In Eq. (2.5), $\bar{\phi}_i$ is the average value of the scalar over the edge and can be calculated as the average of the values of the two nodes that makes up the face or as the average of the value at the two cell centers on either side of the face. It will be shown that using the nodal values adds simplicity to the final equations. $\hat{n}_{xi}$ is the x component of the surface unit-normal vector for face i. The unit-normal is normalized by Γ_i which is the length of face i. A nonhatted n will indicate a nonunit normal vector component, which is found using Eq. (2.6).

$$\begin{aligned} \hat{n}_{xi} &= \frac{n_{xi}}{\Gamma_i} \Rightarrow n_{xi} = \hat{n}_{xi}\Gamma_i \\ \hat{n}_{yi} &= \frac{n_{yi}}{\Gamma_i} \Rightarrow n_{yi} = \hat{n}_{yi}\Gamma_i \end{aligned} \tag{2.6}$$

The normal vectors for each of the three edges are calculated as follows:

$$\begin{aligned} n_{x1} &= y_3 - y_2 \quad & n_{y1} &= -(x_3 - x_2) \\ n_{x2} &= y_1 - y_3 \quad & n_{y2} &= -(x_1 - x_3) \\ n_{x3} &= y_2 - y_1 \quad & n_{y3} &= -(x_2 - x_1) \end{aligned} \tag{2.7}$$

The value of the gradient on each edge will be treated as the average of the value at the two nodes that define the edge (note from Fig. 3 that the edge number corresponds to the node opposite the edge) and will be designated by an overbar.

$$\begin{aligned} \bar{\phi}_1 &= \frac{1}{2}(\phi_2 + \phi_3) \\ \bar{\phi}_2 &= \frac{1}{2}(\phi_1 + \phi_3) \\ \bar{\phi}_3 &= \frac{1}{2}(\phi_1 + \phi_2) \end{aligned}$$

Using the numbering convention from Fig. 3 results in the following equation.

$$\frac{\partial \phi}{\partial x} = \frac{1}{A}\sum_{i=1}^{3} \bar{\phi}_i\, \hat{n}_{xi}\, \Gamma_i = \frac{1}{A}\sum_{i=1}^{3} \bar{\phi}_i\, n_{xi} = \frac{1}{A}\left(\frac{1}{2}(\phi_2 + \phi_3)n_{x1} + \frac{1}{2}(\phi_3 + \phi_1)n_{x2} + \frac{1}{2}(\phi_1 + \phi_2)n_{x3}\right)$$

Rearranging this equation generates Eq. (2.8):

$$\begin{aligned} \frac{\partial \phi}{\partial x} &= \frac{1}{2A}\left((\phi_2 + \phi_3)n_{x1} + (\phi_3 + \phi_1)n_{x2} + (\phi_1 + \phi_2)n_{x3}\right) \\ \frac{\partial \phi}{\partial y} &= \frac{1}{2A}\left((\phi_2 + \phi_3)n_{y1} + (\phi_3 + \phi_1)n_{y2} + (\phi_1 + \phi_2)n_{y3}\right) \end{aligned} \tag{2.8}$$

Rearranging Eq. (2.8) gives the most common form for the gradient component equations.

$$\begin{aligned}\frac{\partial\phi}{\partial x} &= \frac{1}{2A}\left(\left(n_{x2}+n_{x3}\right)\phi_1+\left(n_{x3}+n_{x1}\right)\phi_2+\left(n_{x1}+n_{x2}\right)\phi_3\right)\\ \frac{\partial\phi}{\partial y} &= \frac{1}{2A}\left(\left(n_{y2}+n_{y1}\right)\phi_1+\left(n_{y3}+n_{y1}\right)\phi_2+\left(n_{y1}+n_{y2}\right)\phi_3\right)\end{aligned} \tag{2.9}$$

Using the fact that the normals sum to zero – which can be easily seen from Eq. (2.7) – Eq. (2.9) simplifies to Eq. (2.10), and the derivatives from Eq. (2.10) are used to form the 2D gradient of the scalar, which is shown in Eq. (2.11).

$$\begin{aligned}\frac{\partial\phi}{\partial x} &= \frac{1}{2A}\left(-n_{x1}\phi_1-n_{x2}\phi_2-n_{x3}\phi_3\right)\\ \frac{\partial\phi}{\partial y} &= \frac{1}{2A}\left(-n_{y1}\phi_1-n_{y2}\phi_2-n_{y3}\phi_3\right)\end{aligned} \tag{2.10}$$

$$\nabla\phi=\begin{bmatrix}\frac{\partial\phi}{\partial x}\\ \frac{\partial\phi}{\partial y}\end{bmatrix}=-\frac{1}{2A}\begin{bmatrix}n_{x1}\phi_1+n_{x2}\phi_2+n_{x3}\phi_3\\ n_{y1}\phi_1+n_{y2}\phi_2+n_{y3}\phi_3\end{bmatrix} \tag{2.11}$$

2.2 LAPLACE AND POISSON'S EQUATION

A large segment of this report deals with the Winslow elliptic smoothing equations, which are derived in detail in Appendix B and discussed in further detail in Section 5.0. The Winslow equations are used to ensure smoothness in a computational mesh and are derived by applying the Laplacian (with respect to the physical mesh coordinates) to the computational coordinates. Because this system of equations is based on Laplace's equation (for a homogenous system) or Poisson's equation (for a nonhomogeneous system) it is interesting to compare it with other systems that follow the same laws.

Some other systems where the Laplacian is used are the steady-state heat equation $\left(\nabla^2 T=0\right)$ and the velocity potential equation. When a flow is steady, irrotational, and isentropic, the Euler equations can be simplified into the potential velocity equation, and it can be observed that $\nabla^2\psi=0$ and $\nabla^2\phi=0$ where lines of constant ψ are known as streamlines and lines of constant ϕ are known as equipotential lines. In this situation ϕ is the equipotential function and ψ is the so-called stream function, defined so as to satisfy continuity identically (Ref. 9).

The equipotential function and stream function are developed by looking at the continuity equation for incompressible flow.

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \tag{2.12}$$

The equipotential function, ϕ, is defined such that $\frac{\partial \phi}{\partial x} = u$ and $\frac{\partial \phi}{\partial y} = v$. Plugging these relationships into Eq. (2.12) gives Laplace's equation for ϕ, which is $\nabla^2 \phi = 0$. If the stream function, ψ, is defined such that $u \equiv \frac{\partial \psi}{\partial y}$ and $v \equiv -\frac{\partial \psi}{\partial x}$, then the continuity equation is identically satisfied:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial^2 \psi}{\partial x \partial y} - \frac{\partial^2 \psi}{\partial x \partial y} \equiv 0$$

This relationship can now be plugged into the equation for vorticity (ω_z) and, if the flow is irrotational, the result is again Laplace's equation:

$$-\left(\frac{\partial u}{\partial y} - \frac{\partial v}{\partial x}\right) = \omega_z \tag{2.13}$$

$$\frac{\partial^2 \psi}{\partial y^2} + \frac{\partial^2 \psi}{\partial x^2} = -\omega_z$$

$$\nabla^2 \psi = -\omega_z$$

$$\nabla^2 \psi = 0 \tag{2.14}$$

It is interesting to examine a flowfield generated using the velocity potential equations, such as the one shown in Fig. 4, where the darker blue lines are streamlines, and the lighter blue lines are equipotential lines. The lines of constant ψ and constant ϕ form a system similar to what would be expected when generating a mesh that is smoothed using the elliptic smoothing equations (also based on Laplace's equations).

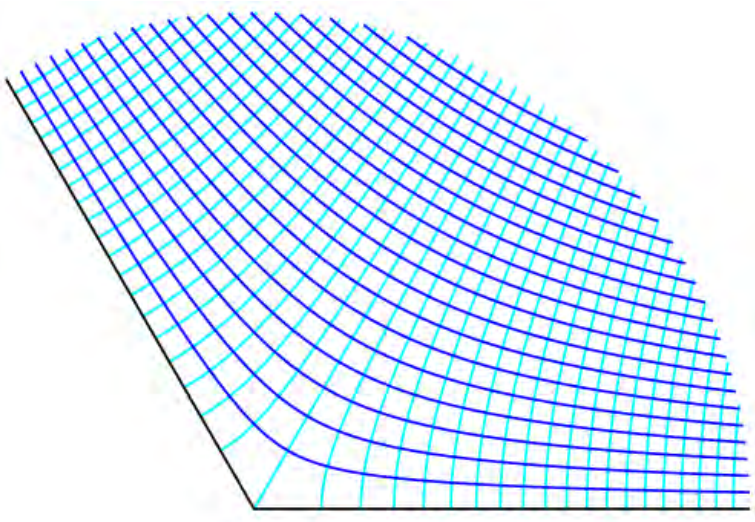

Figure 4. Flowfield Generated Using the Velocity Potential Equation

2.3 LAPLACIAN SMOOTHING VS. WINSLOW SMOOTHING

Prior to work done by Knupp, smoothing techniques on unstructured meshes were dominated by Laplacian smoothing because of its generality and ease of implementation. However, Knupp felt that the additional work of implementing the Winslow elliptic smoothing equations was worth the effort because of the robustness against grid folding that is achieved by using the Winslow equations (Ref. 10). In Laplacian smoothing, node positions are determined by solving the Laplacian of the physical coordinates with respect to the computational coordinates. In two dimensions, this becomes:

$$\begin{aligned} \nabla^2 \mathrm{x} &= \frac{\partial^2 \mathrm{x}}{\partial \xi^2} + \frac{\partial^2 \mathrm{x}}{\partial \eta^2} = 0 \\ \nabla^2 \mathrm{y} &= \frac{\partial^2 \mathrm{y}}{\partial \xi^2} + \frac{\partial^2 \mathrm{y}}{\partial \eta^2} = 0 \end{aligned} \tag{2.15}$$

Laplacian smoothing is easy to implement because the position of a node can be solved as the average of the positions of the N neighboring nodes:

$$\vec{\mathrm{x}}_{\mathrm{n}} = \frac{1}{\mathrm{N}} \sum_{\mathrm{m}=0}^{\mathrm{N}-1} \vec{\mathrm{x}}_{\mathrm{m}} \tag{2.16}$$

Although Laplacian smoothing is easy to implement, its usefulness is limited by the fact that it sometimes results in mesh folding/spillover, and no mathematical guarantee against such folding can be constructed (Ref. 10). The Winslow equations are obtained by requiring that the computational coordinate variables (ξ and η) be harmonic functions and then interchanging the dependent and independent variables in the corresponding Laplace equations. That is, Laplacians of the computational space are now dealt with in respect to physical space, as shown in Eq. (2.17).

$$\begin{aligned} \nabla^2 \xi &= \frac{\partial^2 \xi}{\partial \mathrm{x}^2} + \frac{\partial^2 \xi}{\partial \mathrm{y}^2} = 0 \\ \nabla^2 \eta &= \frac{\partial^2 \eta}{\partial \mathrm{x}^2} + \frac{\partial^2 \eta}{\partial \mathrm{y}^2} = 0 \end{aligned} \tag{2.17}$$

The derivation of the Winslow equations from the above Laplace equations (Eq. [2.17]) is discussed in detail in Appendix B, but it is worth noting that the technique that was used by Knupp was to let a local discrete uniform computational space for a node with a valence count of N be given by Eq. (2.18) and then letting the physical coordinates of the nodes be a function of the computational space coordinates: $\mathrm{x} = \mathrm{x}(\xi, \eta)$ and $\mathrm{y} = \mathrm{y}(\xi, \eta)$.

$$\begin{aligned} \xi_{\mathrm{m}} &= \cos\theta_{\mathrm{m}} \\ \eta_{\mathrm{m}} &= \sin\theta_{\mathrm{m}} \end{aligned} \tag{2.18}$$

where: $\theta = \dfrac{2\pi \mathrm{m}}{\mathrm{N}}$

By assuming that there exist smooth functions, $x = x(\xi,\eta)$ and $y = y(\xi,\eta)$, on the local computational space and that these functions can be approximated about the origin by a Taylor series expansion, the expansion can be used to approximate the first and second derivates of x and y that are needed to construct the necessary coefficients of the Winslow equations.

3.0 ISOTROPIC WINSLOW SMOOTHING

3.1 ELLIPTIC SMOOTHING

Winslow smoothing is the term commonly used to describe the method of elliptic mesh smoothing based on manipulating a mesh (structured or unstructured) using the Winslow elliptic smoothing equations. The Winslow equations, which are derived in detail in Appendix B, are found by taking the Laplacian of the computational space coordinates with respect to physical space.

$$\begin{aligned} \nabla^2 \xi &= 0 \\ \nabla^2 \eta &= 0 \end{aligned} \tag{3.1}$$

The Laplacian follows the Min/Max principle, which says that given a scalar field over a region R that satisfies Laplace's equation, $\nabla^2 \phi = 0$, the value of a nonconstant scalar function, ϕ, cannot attain its maximum or minimum in the interior. This can be deduced from the fact that Laplace's equation, which is said to be harmonic, has the property that the average value over a spherical surface is equal to the value at the center of the sphere (Gauss's harmonic function theorem) (Ref. 11). Suppose that one wishes to solve Laplace's equation in any region R. Consider any point p inside R and a circle of any radius r_0 (such that the circle is inside R). Let the value on the circle be $f(\theta)$. If Laplace's equation holds, the value at any point is the average of the values along any circle of radius r (lying inside R) centered at that point. The proof is by contradiction. Suppose the maximum or minimum was at point p as illustrated in Fig. 5. The value at point p should be the average of all points on any surrounding circle, such as the one with radius r_0; it is impossible for the value at p to be larger or smaller than the maximum or minimum surrounding point.

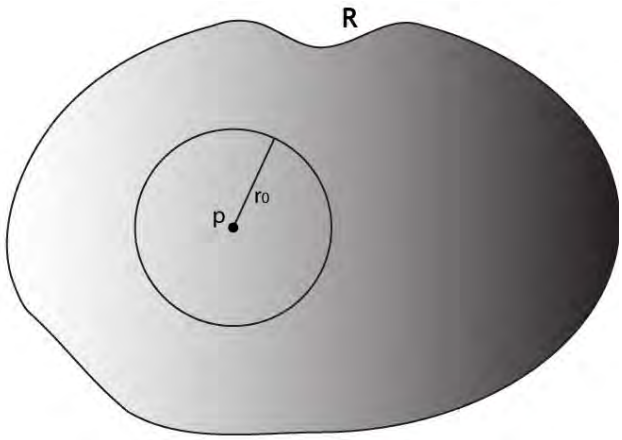

Figure 5. Region Surrounding a Control Volume

The Winslow equations are derived by transforming the Laplacian of the computational space coordinates with respect to physical coordinates such that the equations are now with respect to

computational coordinate and the physical coordinates are now the dependent variables. This transformation is necessary because it is generally the physical coordinates that will be modified using elliptic smoothing. The coordinates of the computational space need to be defined (either implicitly or explicitly) and can then be further manipulated to affect the physical mesh as required. The Winslow equations are constructed such that the dependent variables are now the physical coordinates and the partial derivatives are now with respect to the computational coordinates. This is useful because it is the physical coordinates that will be modified using elliptic smoothing. The Winslow equations, in the form most relevant to the work performed herein, are shown below as Eq. (3.2) with the coefficients shown as Eq. (3.3).

$$\begin{aligned} \alpha \frac{\partial^2 \mathrm{x}}{\partial \xi^2} - 2\beta \frac{\partial^2 \mathrm{x}}{\partial \xi \partial \eta} + \gamma \frac{\partial^2 \mathrm{x}}{\partial \eta^2} = 0 \\ \alpha \frac{\partial^2 \mathrm{y}}{\partial \xi^2} - 2\beta \frac{\partial^2 \mathrm{y}}{\partial \xi \partial \eta} + \gamma \frac{\partial^2 \mathrm{y}}{\partial \eta^2} = 0 \end{aligned} \tag{3.2}$$

$$\begin{aligned} \alpha &= \mathrm{x}_\eta{}^2 + \mathrm{y}_\eta{}^2 \\ \beta &= \mathrm{x}_\xi \mathrm{x}_\eta + \mathrm{y}_\xi \mathrm{y}_\eta \\ \gamma &= \mathrm{x}_\xi{}^2 + \mathrm{y}_\xi{}^2 \end{aligned} \tag{3.3}$$

3.2 VIRTUAL CONTROL VOLUMES

In order to explore the computational space of a two dimensional unstructured mesh, consider a small subsection of a mesh comprised of triangles such as that shown in Fig. 6. There are three internal nodes (marked as: ●●●). Unlike structured grid methods, unstructured methods have not traditionally had a well-defined computational space associated with the physical space. While a global unstructured computational space remains elusive, nodes in an unstructured mesh can be mapped to an individualized computational space based on virtual control volumes decoupled from one another in computational space. (They remain loosely coupled through their relationship in physical space.) The computational space that is suggested by Karman et al (Ref. 12) is one in which the node in question lies at the center of a virtual control volume based on a regular (equiangular) polygon whose vertices intersect a unit circle. For each of the internal nodes shown in physical space in Fig. 6, the corresponding computational space can be seen in Fig. 7.

Defining the computational space in this way lends itself well to solving the Winslow equations (especially in inviscid regions where an isotropic mesh is desired) and allows the equations to be solved even if the physical grid is inextricably tangled. The computational space (ξ–η space) shown in Fig. 7 is based on the valence count of the node in physical space, which is mapped to the center of a regular polygon whose vertices intersect a unit circle. The nodes in this figure are color-coordinated to the nodes in physical space in Fig. 6.

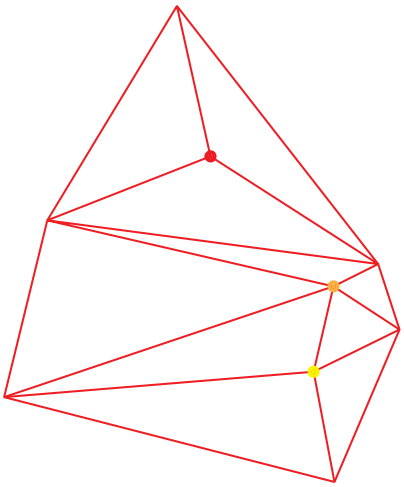

Figure 6. Subsection of an Unstructured Mesh

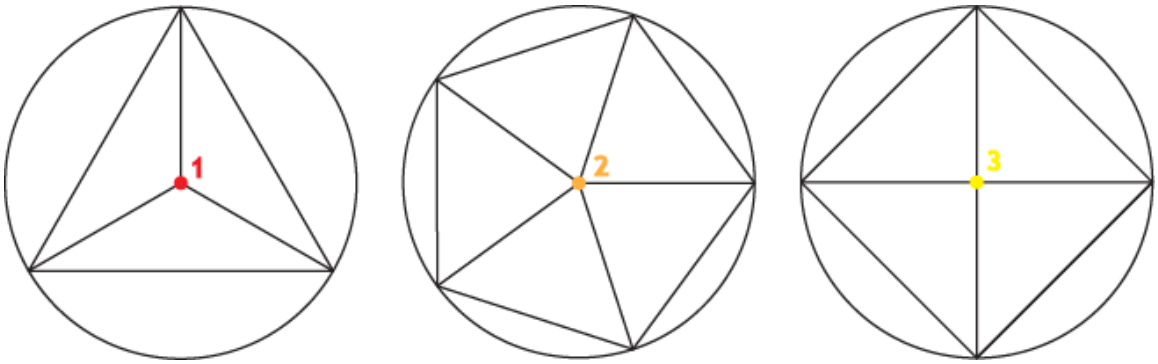

Figure 7. Virtual Control Volumes in Computational Space

3.3 DISCRETE FORM OF WINSLOW EQUATIONS

The discretization of the Winslow equations focuses on the Laplace form of the equations (i.e., the homogeneous equations without any forcing functions). The justification for this is that a desired grid spacing that might incline a user to use the Poisson form of the equations can also be generated by manipulating the computational space, so forcing functions become superfluous. The Laplace form of the Winslow equations for x and y (using condensed notation for the derivatives) are $\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} = 0$ and $\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} = 0$.

The logic described in this section is used to get the Winslow equations in a form that can be used to solve the overall global system consisting of all nodes in a region that is to be smoothed. (Note that the physical coordinate in the x direction is used in the following derivation but that the derivation also applies to the gradients of y with respect to ξ and η as well.)

Because the Winslow equations are cast in computational space, the gradient operator (∇) will be defined such that when operating (through the dot product) on a vector in computational space the results are as shown with Eqs. (3.4) and (3.5).

$$\vec{F} = \begin{bmatrix} x_\xi \\ x_\eta \end{bmatrix} \Rightarrow \nabla \cdot \vec{F} = \frac{\partial}{\partial \xi} x_\xi + \frac{\partial}{\partial \eta} x_\eta \tag{3.4}$$

$$\begin{aligned} \vec{F} &= \begin{bmatrix} x_\xi \\ 0 \end{bmatrix} \Rightarrow \nabla \cdot \vec{F} = \frac{\partial}{\partial \xi}\left(x_\xi\right) + 0 = x_{\xi\xi} \Rightarrow \vec{F} \cdot \hat{n} = x_\xi \hat{n}_\xi \\ \vec{F} &= \begin{bmatrix} x_\eta \\ 0 \end{bmatrix} \Rightarrow \nabla \cdot \vec{F} = \frac{\partial}{\partial \xi}\left(x_\eta\right) + 0 = x_{\xi\eta} \Rightarrow \vec{F} \cdot \hat{n} = x_\eta \hat{n}_\xi \\ \vec{F} &= \begin{bmatrix} 0 \\ x_\eta \end{bmatrix} \Rightarrow \nabla \cdot \vec{F} = 0 + \frac{\partial}{\partial \eta}\left(x_\eta\right) = x_{\eta\eta} \Rightarrow \vec{F} \cdot \hat{n} = x_\eta \hat{n}_\eta \end{aligned} \tag{3.5}$$

The Winslow equations in a two-dimensional control volume can be written in integral form as shown as Eq. (3.6). The logic illustrated in Eq. (3.5) combined with the divergence theorem from Section 2 generates the homogeneous Winslow equations in terms of the surface integral over the surface of the control volume. This is shown as Eq. (3.7):

$$\begin{aligned} \iint_A \left(\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta}\right) dA = 0 \\ \iint_A \left(\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}\right) dA = 0 \end{aligned} \tag{3.6}$$

$$\begin{aligned} \oint \left(\alpha x_\xi \hat{n}_\xi - 2\beta x_\eta \hat{n}_\xi + \gamma x_\eta \hat{n}_\eta\right) dS = 0 \\ \oint \left(\alpha y_\xi \hat{n}_\xi - 2\beta y_\eta \hat{n}_\xi + \gamma y_\eta \hat{n}_\eta\right) dS = 0 \end{aligned} \tag{3.7}$$

The virtual control volume for a node with a valence count of four, such as the node represented as ● in Fig. 6, is shown in Fig. 8. The vectors t_1 through t_4 in Fig. 8 are the nonunit normal vectors for the surface of the overall virtual control volume. As will become clear when the equations are further expanded, t is used instead of the more typical n when describing the overall surface normal vectors because these vectors need to be distinguished from the normal vectors of the individual triangular elements that comprise the control volume (which are used to calculate the gradients in the control volume).

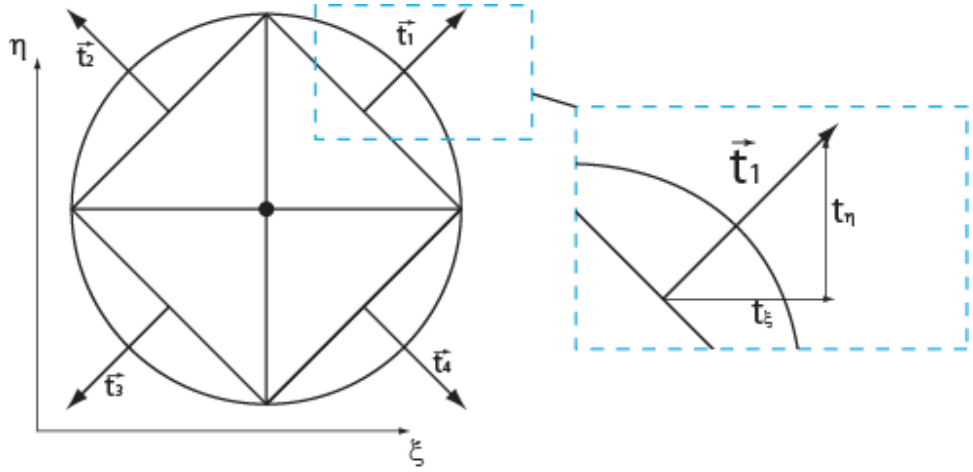

Figure 8. Virtual Control Volume for a Valence-Count 4 Node

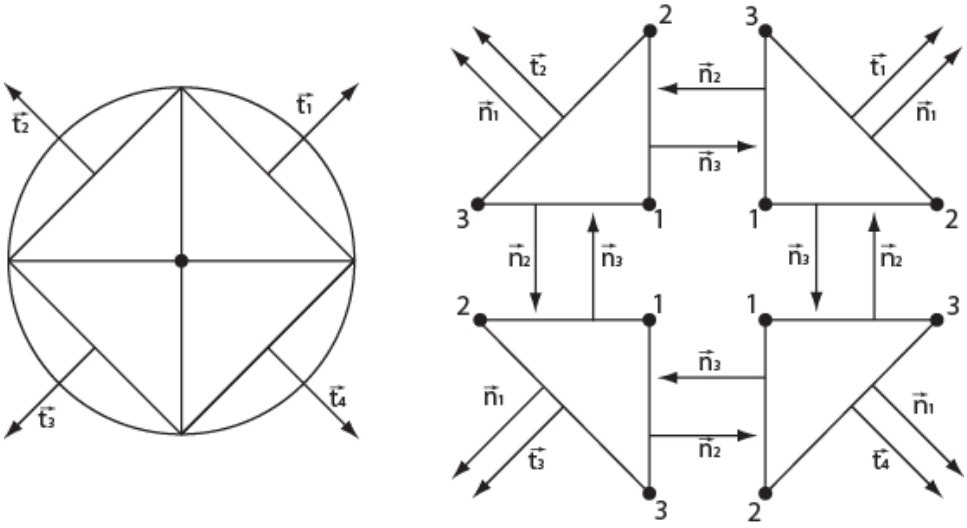

Figure 9. Control Volume for a Single Valence-4 Node Exploded Into its Individual Triangular Components

In Fig. 9, the control volume from Fig. 8 is shown exploded into the triangular sectors that make up the overall control volume. The nonunit normal vectors for each of the triangles comprising a control volume are illustrated in Fig. 9. The variable t is the nonunit surface normal vector of the overall control volume (i.e., $t_i = \hat{t}_i \, d\Gamma$ where $\hat{t}_i$ is the unit normal vector and $d\Gamma$ is the length of face i). The discretized Winslow equation (for x) defined for a given control volume with a valence count of N (i.e., N is the number of neighboring nodes and thus the number of triangles comprising the control volume) is shown below as Eq. (3.8).

$$\iint \left(\alpha x_\xi \hat{n}_\xi - 2\beta x_\eta \hat{n}_\xi + \gamma x_\eta \hat{n}_\eta \right) dS = \sum_{i=1}^{N} \left[\alpha x_\xi \hat{t}_\xi - 2\beta x_\eta \hat{t}_\xi + \gamma x_\eta \hat{t}_\eta \right]_i d\Gamma = 0$$

$$\sum_{i=1}^{N} \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_i = 0 \tag{3.8}$$

Expanding this summation for a node with a valence count of 4, such as the central node shown in Figs. 8 and 9, gives:

$$\sum_{i=1}^{4} \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_i = 0$$

$$\left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_1 + \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_2$$
$$+ \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_3 + \left[\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta \right]_4 = 0$$

The discretized integral expanded above is the integral for the central node's entire control volume, whose surface has been discretized into four line segments. However, the elliptic smoothing equations apply to an arbitrary control volume, so the argument can be made that the only way for this integral to be zero in general is if the integrand is zero. Another way to

think about this is that the discrete form of the Winslow equations has to apply everywhere in the domain, both globally and locally. Thus, for each of the sectors (the individual triangles in discretized space) of the control volume, the following equations apply:

$$\begin{aligned}\alpha x_\xi t_\xi - 2\beta x_\eta t_\xi + \gamma x_\eta t_\eta = 0\\ \alpha y_\xi t_\xi - 2\beta y_\eta t_\xi + \gamma y_\eta t_\eta = 0\end{aligned} \tag{3.9}$$

The Gradient equations from Section 2 can be applied to get the values of the gradients (with respect to the computational coordinates) in each triangle.

$$\begin{aligned}x_\xi &= -\frac{1}{2A}\left(n_{\xi 1}x_1 + n_{\xi 2}x_2 + n_{\xi 3}x_3\right)\\ x_\eta &= -\frac{1}{2A}\left(n_{\eta 1}x_1 + n_{\eta 2}x_2 + n_{\eta 3}x_3\right)\end{aligned} \tag{3.10}$$

$$\begin{aligned}y_\xi &= -\frac{1}{2A}\left(n_{\xi 1}y_1 + n_{\xi 2}y_2 + n_{\xi 3}y_3\right)\\ y_\eta &= -\frac{1}{2A}\left(n_{\eta 1}y_1 + n_{\eta 2}y_2 + n_{\eta 3}y_3\right)\end{aligned} \tag{3.11}$$

In the equations for the gradients, $n_{\xi 1}$ $n_{\xi 2}$ and $n_{\xi 3}$ represent the ξ component of the nonunit normal vector of the three sides of the triangle and likewise for η. Using the gradient relationships from Eqs. (3.10) and (3.11) in Eq. (3.9) and multiplying by -2A (which can be done because no forcing functions are being used and the equations are homogeneous) results in Eq. (3.12).

$$\begin{aligned}&\alpha\left(n_{\xi 1}x_1 + n_{\xi 2}x_2 + n_{\xi 3}x_3\right)t_\xi - 2\beta\left(n_{\eta 1}x_1 + n_{\eta 2}x_2 + n_{\eta 3}x_3\right)t_\xi\\ &+\gamma\left(n_{\eta 1}x_1 + n_{\eta 2}x_2 + n_{\eta 3}x_3\right)t_\eta = 0\\ &\alpha\left(n_{\xi 1}y_1 + n_{\xi 2}y_2 + n_{\xi 3}y_3\right)t_\xi - 2\beta\left(n_{\eta 1}y_1 + n_{\eta 2}y_2 + n_{\eta 3}y_3\right)t_\xi\\ &+\gamma\left(n_{\eta 1}y_1 + n_{\eta 2}y_2 + n_{\eta 3}y_3\right)t_\eta = 0\end{aligned} \tag{3.12}$$

Rearranging Eq. (3.12) gives the Winslow equations in a form that allows them to be solved for the central node (x_1,y_1) using an iterative method such as the Point Implicit method. This final form is shown as Eq. (3.13).

$$\begin{aligned}&x_1\left(\alpha n_{\xi 1}t_\xi - 2\beta n_{\eta 1}t_\xi + \gamma n_{\eta 1}t_\eta\right) + x_2\left(\alpha n_{\xi 2}t_\xi - 2\beta n_{\eta 2}t_\xi + \gamma n_{\eta 2}t_\eta\right)\\ &+x_3\left(\alpha n_{\xi 3}t_\xi - 2\beta n_{\eta 3} + \gamma n_{\eta 3}t_\eta\right) = 0\\ &y_1\left(\alpha n_{\xi 1}t_\xi - 2\beta n_{\eta 1}t_\xi + \gamma n_{\eta 1}t_\eta\right) + y_2\left(\alpha n_{\xi 2}t_\xi - 2\beta n_{\eta 2}t_\xi + \gamma n_{\eta 2}t_\eta\right)\\ &+y_3\left(\alpha n_{\xi 3}t_\xi - 2\beta n_{\eta 3}t_\xi + \gamma n_{\eta 3}t_\eta\right) = 0\end{aligned} \tag{3.13}$$

The values in parentheses in Eq. (3.13) can be thought of as weights for the Winslow equations at each node, and the two equations shown in Eq. (3.13) can be rewritten in matrix form as follows:

$$x_1 w_{11} + x_2 w_{12} + x_3 w_{13} = 0$$
$$y_1 w_{21} + y_2 w_{22} + y_3 w_{23} = 0$$

$$\begin{bmatrix} w_{11} & 0 \\ 0 & w_{21} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + \begin{bmatrix} w_{12} & 0 \\ 0 & w_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} + \begin{bmatrix} w_{13} & 0 \\ 0 & w_{23} \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = 0 \tag{3.14}$$

To solve the overall system, a global coefficient matrix is constructed where each element of the global matrix is a 2×2 matrix and each row of the global matrix represents a single node in the system. The values of the elements of the row associated with a given node are constructed by using the values of w_{11} and w_{21} as contributions to the diagonal element of the row (the central node) and the values of w_{12}, w_{22}, w_{13}, and w_{23} as contributions to the off-diagonal elements. Refer to Fig. 9 and it becomes clear that the contributions for each triangle making up the control volume for a node are being summed to satisfy the integrals first shown as Eq. (3.7) and then discretized in Eq. (3.8).

A node with a valence count of 4, such as the central node shown in Fig. 9, will have a control volume comprised of 4 triangles, each of which will have a vertex that represents the central node and two vertices that represent two of the central node's neighbors. The local numbering convention used for each of the triangles is that the central node is node number 1 and the two neighboring nodes are numbered node 2 and node 3 based on the right-hand rule. The weights for each of the nodes in each of the triangles are calculated based on the normal vectors and the Winslow coefficients for each triangle. (The gradients, and thus the coefficients, are treated as constant within each triangle.) When the contributions for each of the triangles have been summed up, the off-diagonal contributions (w_{12}, w_{22}, w_{13}, and w_{23}) can be moved to the right-hand side of the global equation, and central node values (w_{11} and w_{21}), which are on the diagonal, can be solved in an iterative fashion.

3.4 ISOTROPIC WINSLOW EQUATIONS APPLIED TO FLAT PLATE

A two-dimensional flat plate mesh was created using the Gridgen software package (Ref. 13). This mesh (shown in Fig. 10) closely resembled a mesh described in Ref. 14 which explored the area of mesh movement but used entirely different techniques that were based on the Linear Elastic equations. Three cases were examined using the mesh from Fig. 10: rotation, translation, and a warping of the flat plate surface. The results from these three cases are shown in Fig. 11. For each of the cases in Fig. 11, the mesh is shown on the left after the inner surface is moved but before smoothing has been performed, and then on the right after smoothing has been performed.

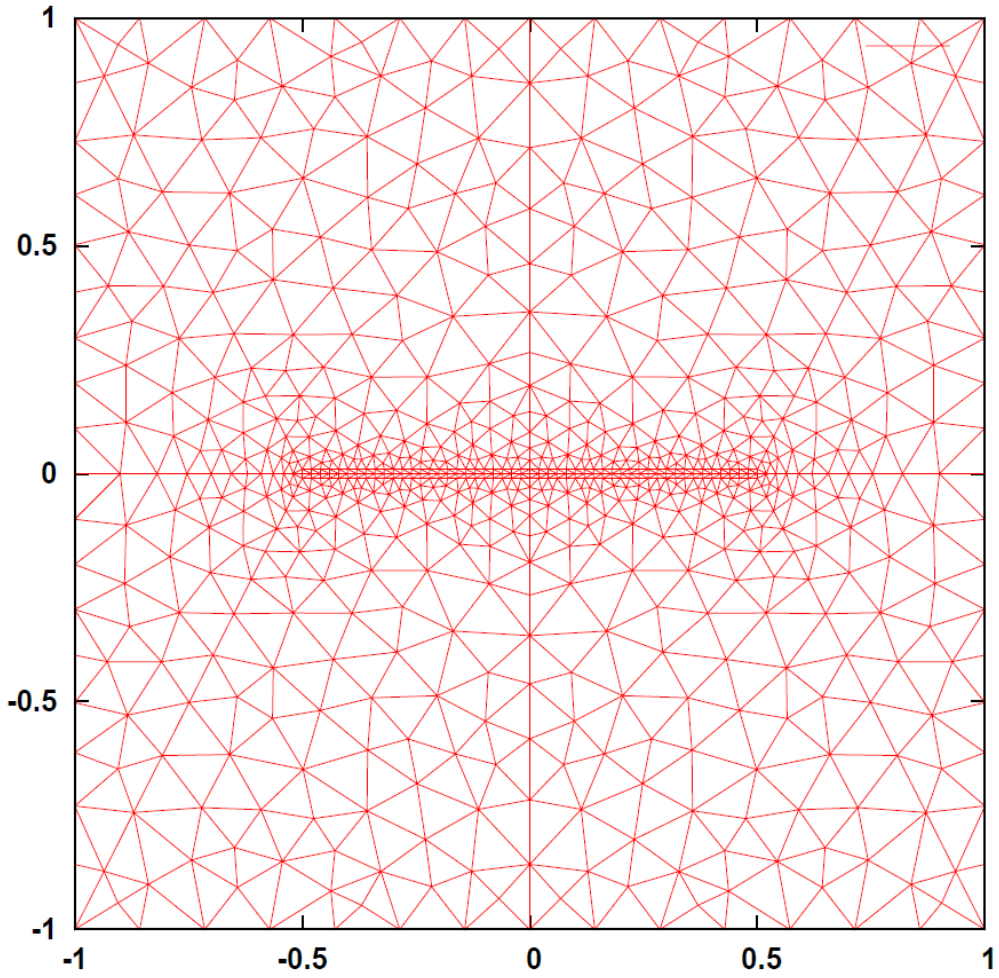

Figure 10. Refined Symmetric Flat Plate Mesh

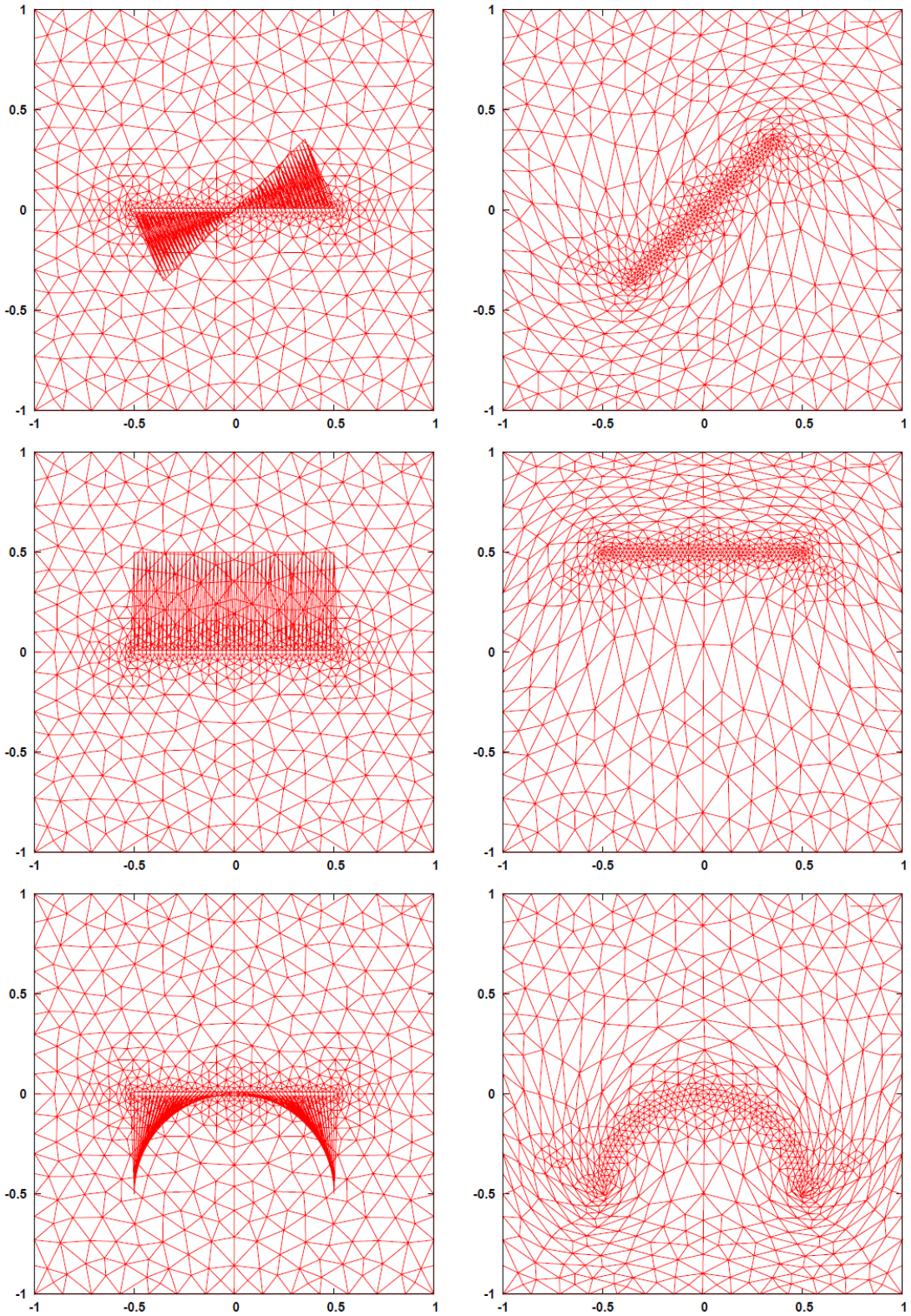

Figure 11. Results from Applying Winslow Equations to Flat Plate Mesh That has Been Rotated, Translated, and Warped into a Semi-Circle

Although all of the grids in Fig. 11 look relatively good, the mesh where the flat plate has been warped into a semi-circle illustrates the need to have the ability to move boundary points if a surface near another boundary is going to be modified. The mesh near the right endpoint of the flat plate is shown in Fig. 12 and again in a close-up view in Fig. 13. The close-up in Fig. 13 follows two sets of points (marked as black and blue dots) before and after mesh movement is performed and shows the difficulty in trying to wrap the mesh around the new geometry while retaining the original connectivity and outer surface grid spacing.

In Fig. 13 it can be seen that the smoothing equations attempt to wrap the two lines of connectivity around the warped flat plate, but in some areas the spacing and limited connectivity between the two lines in not adequate to generate a high-quality mesh. Before the flat plate is warped, there is adequate space between the two sets of nodes to accommodate the connectivity between the sets. However, after the flat plate is warped and the mesh has been smoothed, the two sets of nodes are driven together in a way that makes it difficult for valid connectivity to be maintained.

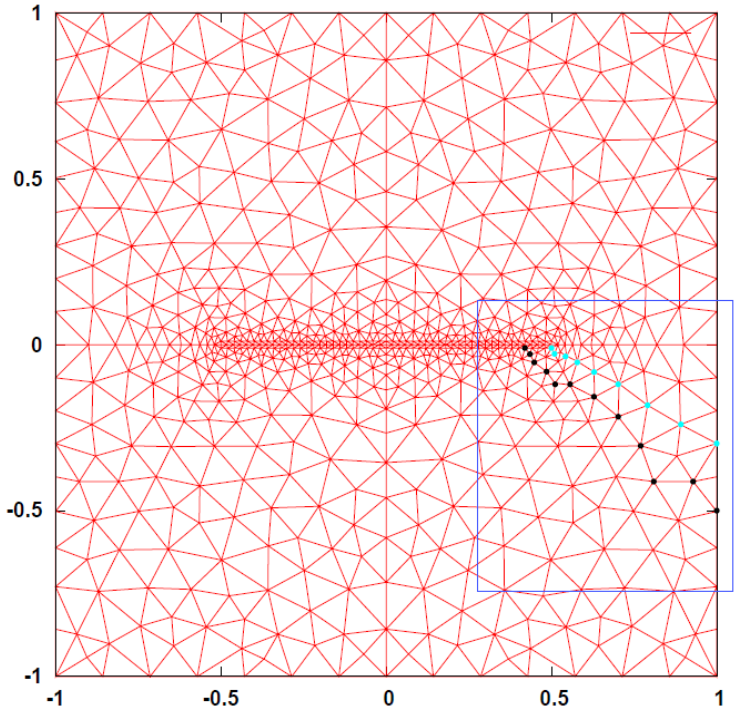

Figure 12. Lines of Connectivity Between an Inner Boundary and an Outer Boundary

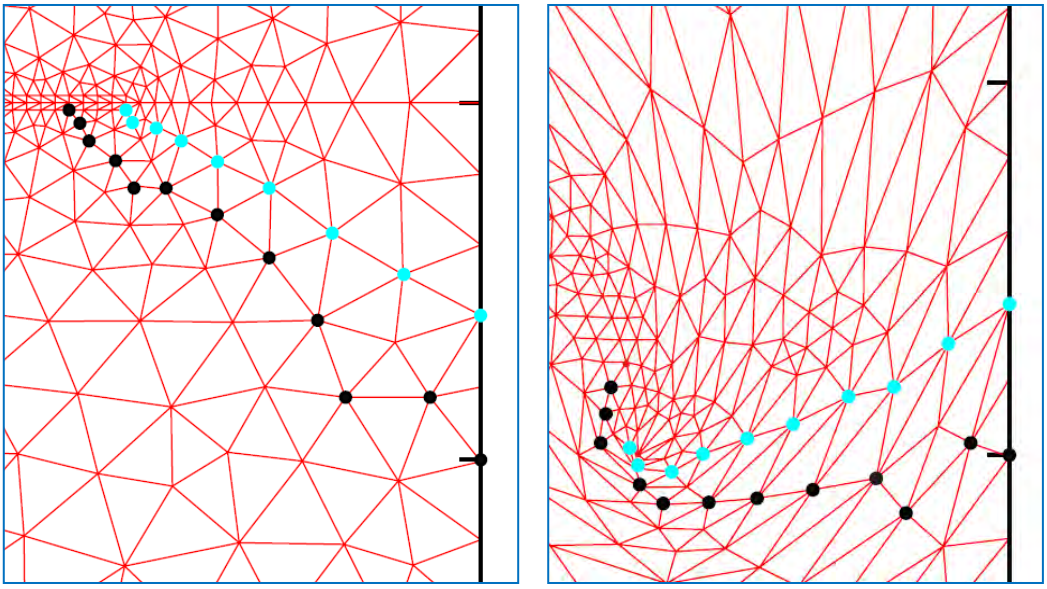

Figure 13. Connectivity Path Between Surface and Outer Boundary Before and After Surface Modification

3.5 ISOTROPIC WINSLOW EQUATIONS APPLIED TO NACA0012 AIRFOIL

It should be noted that although isotropic Winslow equations have significant limitations, there are also many situations in which they are extremely valuable tools for mesh manipulation. One of these situations is to smooth the inviscid mesh of a body that has been moved or deformed in some manner. Another situation where the isotropic Winslow equations can add significant value is in adaptive mesh refinement where h-refinement is employed.

A NACA0012 airfoil is used to illustrate the implementation and value of the isotropic Winslow grid smoothing algorithms. The airfoil, which had an initial angle of attack of 0 deg, was rotated to a nose-down position of 60 deg and the mesh was smoothed. The grid before and after the rotation can be seen in Figs. 14 and 15. Note that after the initial rotation, the mesh is completely invalid. The boundary points associated with the airfoil surface were moved, but all of the interior points remained stationary, which resulted in a mesh containing extreme grid crossing and skewness. This is not an issue when structured grids are utilized because structured grids have an implicit computational space that is not invalidated by an invalid physical mesh as long as the grid connectivity does not change. Likewise, by using equal angle/equal edge-length virtual control volume as the computational space for the unstructured mesh, the invalid physical space was not an issue, and a high-quality inviscid mesh was generated for the transformed airfoil mesh.

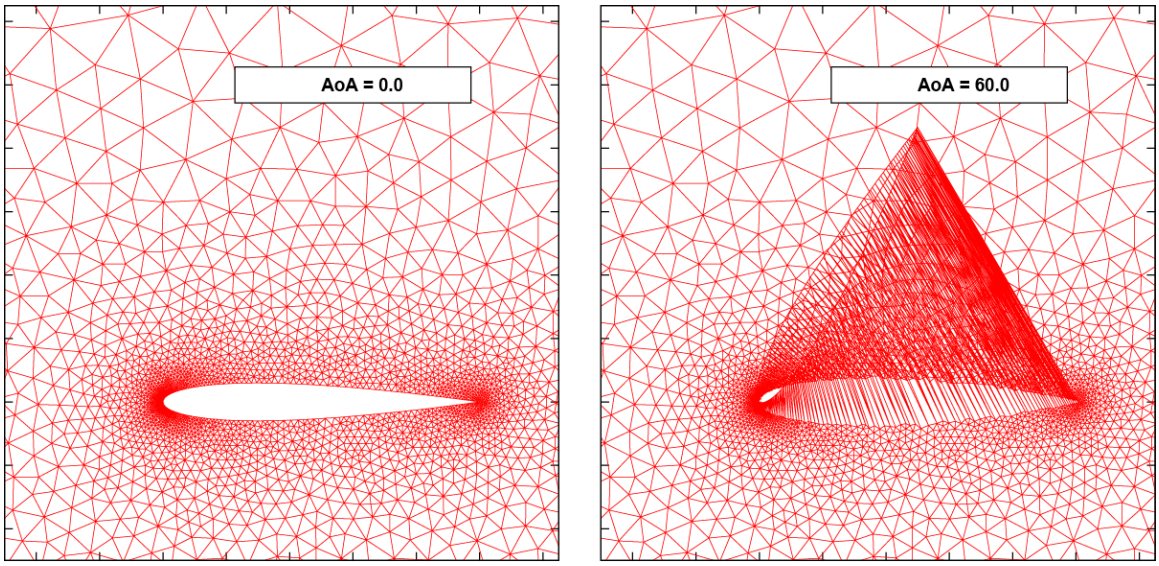

Figure 14. NACA0012 Airfoil Before and After Rotation

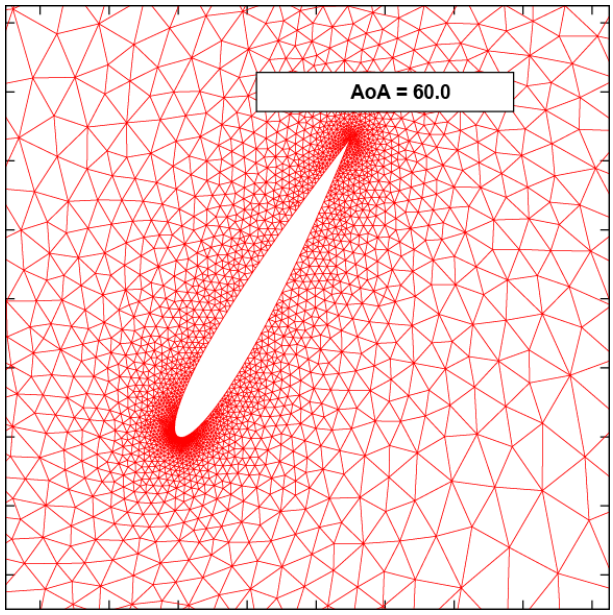

Figure 15. NACA0012 Mesh After Winslow Smoothing has Been Performed

The next case that is examined is a transonic case involving a NACA0012 airfoil in a Mach 0.95 flow. This is the case that was discussed in the Refinement Section. At Mach 0.95 and zero angle of attack, the airfoil exhibits a distinct sonic line or “fishtail shock” in its wake. The objective of examining this case was to use solution adaptation to adapt the original grid (shown in Section 3) with the hope of capturing the shockwaves coming off the airfoil and also the secondary shock behind it. This was an excellent case for employing the inviscid Winslow equations because the areas of interest (the sonic lines) were in the inviscid region of the mesh and thus an isotropic mesh in these regions was adequate. H-refinement schemes determine regions of the grid where the solution is underresolved and add elements locally to improve the resolution. However, if no smoothing is employed, the mesh can become extremely skewed as

more points are added to a given area. Winslow elliptic smoothing can alleviate this tendency toward skewed cells as additional nodes are added.

Figure 16 shows that the adaptation algorithm, coupled with the Winslow elliptic smoothing algorithm, successfully captured the shock coming off the airfoil and also the secondary shock behind it. As the solution was repeatedly refined, the mesh elements in the area of the shock would have become increasingly skewed if no smoothing was employed. It is likely that eventually the skewness of the cells would have negatively affected the CFD solution. An enlarged image of the final mesh is shown in Fig. 16, and it can be seen that even after many refinement iterations, the mesh elements in the region of the shock still have good isotropic aspect ratios due to the application of isotropic Winslow smoothing.

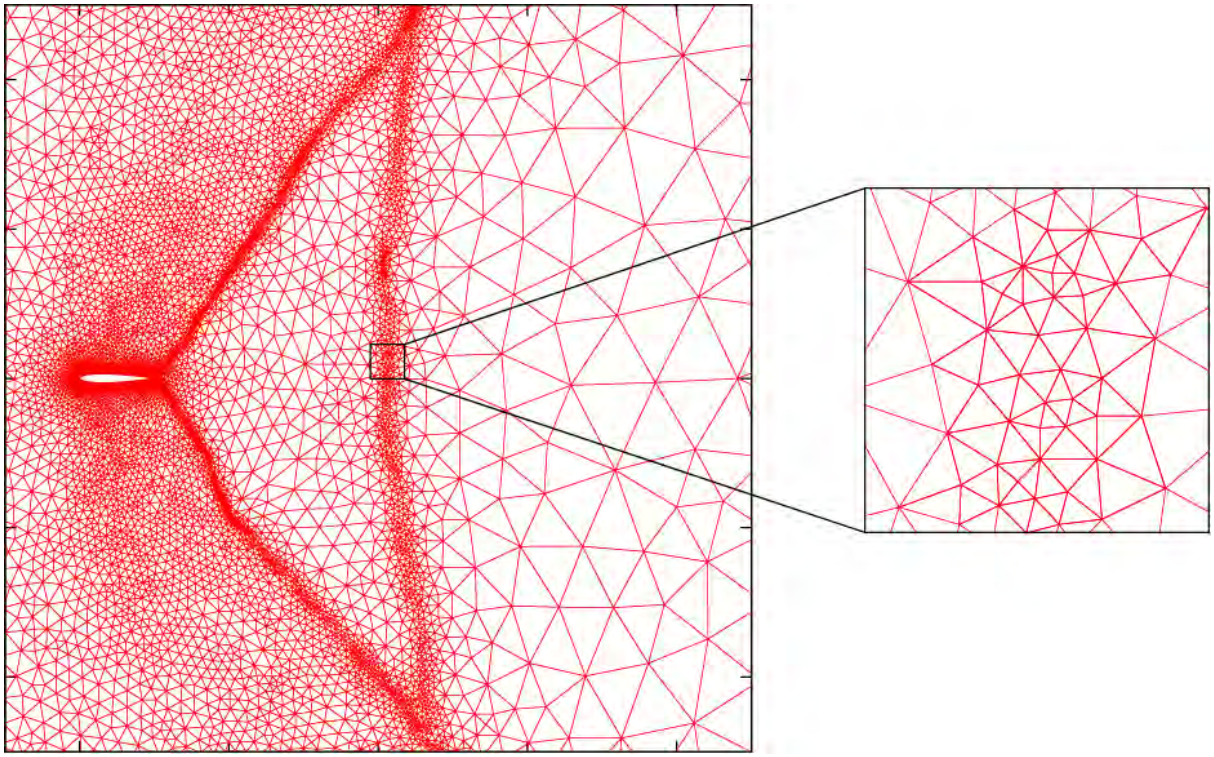

Figure 16. Close-Up of an Isotropic Refined Mesh in the "Fishtail Shock" Region After Six Refinement Iterations

4.0 WINSLOW EQUATIONS ON BOUNDARIES

As noted in the previous section, the Winslow elliptic smoothing equations are derived from Laplace's equations for a parameter distribution over a region. When applied to an unstructured mesh, the equations allow interior mesh points to be moved and smoothed to conform to a moving mesh. Traditionally, the Winslow equations have been applied only to interior mesh points on unstructured meshes. The methodology outlined in this section allows for the implementation of the Winslow equations on boundary points as well. The Winslow equations deal with derivatives of physical space (x,y) with respect to computational space (ξ,η). The equations, which were first defined in Section 5, are restated below.

$$\alpha\frac{\partial^2 x}{\partial\xi^2}-2\beta\frac{\partial^2 x}{\partial\xi\,\partial\eta}+\gamma\frac{\partial^2 x}{\partial\eta^2}=0$$
$$\alpha\frac{\partial^2 y}{\partial\xi^2}-2\beta\frac{\partial^2 y}{\partial\xi\,\partial\eta}+\gamma\frac{\partial^2 y}{\partial\eta^2}=0$$

The Winslow equations are applied to a control volume (in computational space) around each point in the mesh. By employing the Divergence Theorem, a set of surface integrals, shown below, is generated on which the Winslow equations can be solved to smooth the physical coordinates of the mesh points.

$$\iint\left(\alpha\frac{\partial^2 x}{\partial\xi^2}-2\beta\frac{\partial^2 x}{\partial\xi\partial\eta}+\gamma\frac{\partial^2 x}{\partial\eta^2}\right)dA=\oint\left(\alpha\frac{\partial x}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial x}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial x}{\partial\eta}\hat{n}_\eta\right)dS=0$$
$$\iint\left(\alpha\frac{\partial^2 y}{\partial\xi^2}-2\beta\frac{\partial^2 y}{\partial\xi\partial\eta}+\gamma\frac{\partial^2 y}{\partial\eta^2}\right)dA=\oint\left(\alpha\frac{\partial y}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial y}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial y}{\partial\eta}\hat{n}_\eta\right)dS=0$$

In the above equations, $\hat{n}_\xi$ is the ξ component of the unit normal vector on the surface of the control volume in computational space, and $\hat{n}_\eta$ is the η component of the unit normal vector on the surface. The surface integrals can be discretized on the surface as follows:

$$\sum\left(\alpha\frac{\partial x}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial x}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial x}{\partial\eta}\hat{n}_\eta\right)d\Gamma=0$$
$$\sum\left(\alpha\frac{\partial y}{\partial\xi}\hat{n}_\xi-2\beta\frac{\partial y}{\partial\eta}\hat{n}_\xi+\gamma\frac{\partial y}{\partial\eta}\hat{n}_\eta\right)d\Gamma=0 \qquad (4.1)$$

In the discretized equations, $d\Gamma$ is the length of a discrete surface segment. Eq. (4.1) can be simplified by applying the relationship between the unit normal vector ($\hat{n}$) and a nonunit normal vector (n) on a given surface segment. Applying this relationship, which is $\hat{n}\Gamma=n$, simplifies Eq. (4.1) to Eq. (4.2).

$$\sum\left(\alpha\frac{\partial x}{\partial\xi}n_\xi-2\beta\frac{\partial x}{\partial\eta}n_\xi+\gamma\frac{\partial x}{\partial\eta}n_\eta\right)=0$$
$$\sum\left(\alpha\frac{\partial y}{\partial\xi}n_\xi-2\beta\frac{\partial y}{\partial\eta}n_\xi+\gamma\frac{\partial y}{\partial\eta}n_\eta\right)=0 \qquad (4.2)$$

The 9-point unstructured mesh shown in Fig. 17 gives an example of a control volume on which smoothing equations could be applied. In this figure the nodes are numbered in order to illustrate the mapping from physical space to computational space, where the Winslow equations are applied.

The only interior node in Fig. 17 is node 8, and the computational space for node 8 is shown on the right-hand side of Fig. 17. The computational stencil for the interior point (node 8) is relatively straightforward, but no intrinsically obvious stencil exists for a point on a boundary, such as node 5. Recall from Section 1 that the computational stencil of a boundary point is incomplete. In order to complete the computational stencil for a boundary point, the concept of

ghost nodes is introduced. If, for example, it was desired that node 5 in Fig. 17 be allowed to float, a ghost point could be placed as shown in Fig. 18. The ghost point, which in this case is node 9, allows the control volume in computational space to be closed. The closed virtual control volume making up the computational space for the surface node (node 5) can be seen in Fig. 18.

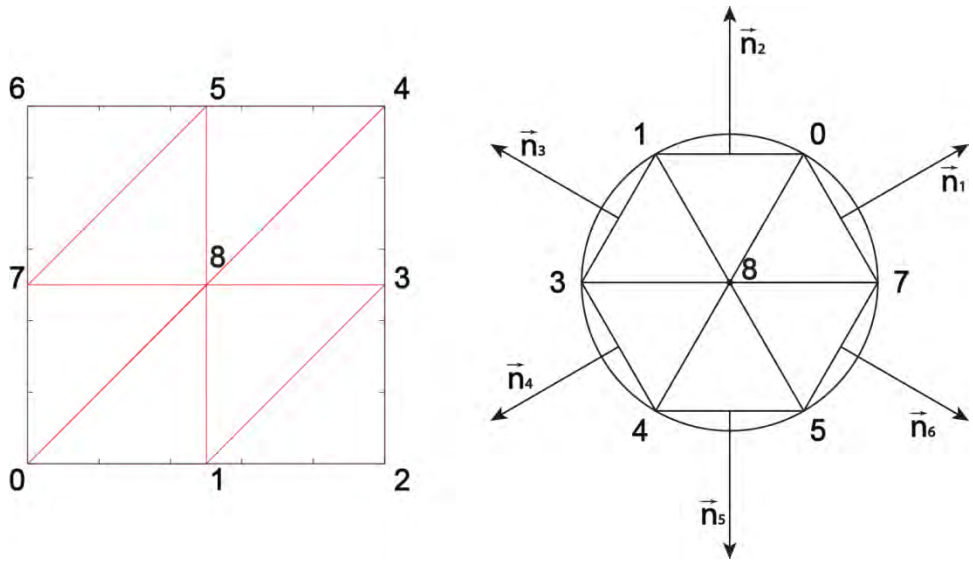

Figure 17. Interior Node (Node 8) in Physical (Left) and Computational (Right) Space

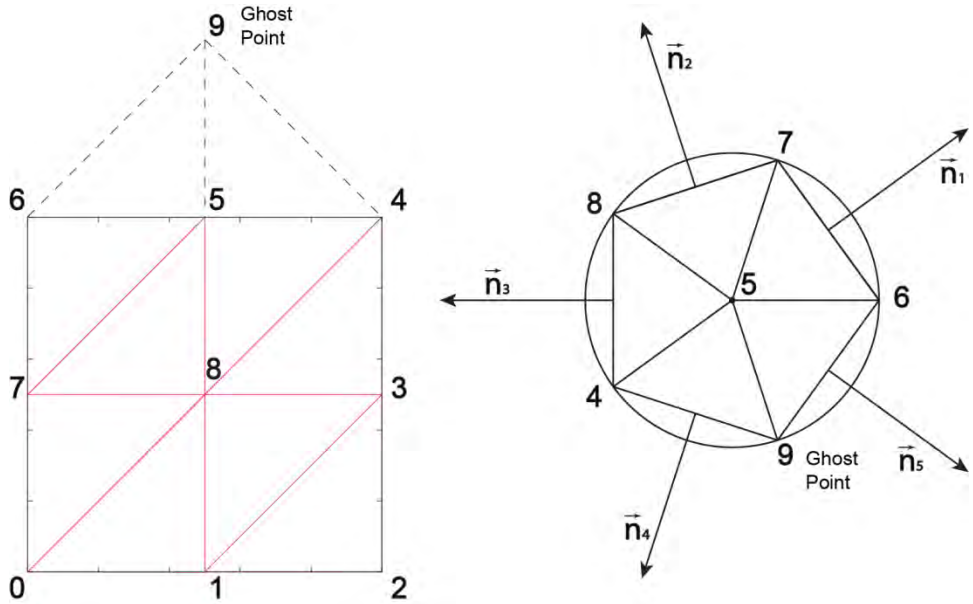

Figure 18. Surface Node (Node 5) and Ghost Node (Node 9) in Physical (Left) and Computational (Right) Space

4.1 PLACING GHOST POINTS

If a node on a surface, such as node 5 in Fig. 18, is to be allowed to float, it will need an associated ghost point (such as node 9 in the previous figures) in order to close the virtual control volume to which the node is mapped in computational space. The Winslow equations

are solved in computational space, and the parameters they solve for are the physical x and y coordinates of node 5 based on the physical x and y values of the connected neighboring nodes (nodes 6, 7, 8, 4, and 9). Because the value of x and y for node 5 is a function of the values of x and y for the connected nodes, it is necessary to place the ghost point at some reasonable location in physical space.

The first attempt at placing the ghost point was to place it at the same location as the associated boundary node. However, this had a fatal effect on the boundary points. For example, when the top outer boundary of a mesh surrounding a flat plate was allowed to float and the ghost point values were set to the values of their associated boundary nodes, the surface points were all driven either to the corners of the boundary or, because of the symmetry characteristics of the mesh, to the symmetry plane. This effect can be seen in Fig. 19.

Upon further consideration, it makes sense that this method of placing ghost points would cause problems. To illustrate this, consider the control volume shown in Fig. 18, which is in computational space. It is apparent from Fig. 18 that node 5 will never be able to reach equilibrium with the surrounding points if one of the surrounding points is always being driven to the same value as the point at the center. This is what is happening as the attempt is made to generate a solution on the flat plate mesh. The central nodes in computational space that represent floating-node boundary points will never be able to reach a converged solution, and nodes in physical space will be driven in some direction until the solver runs out of iterations.

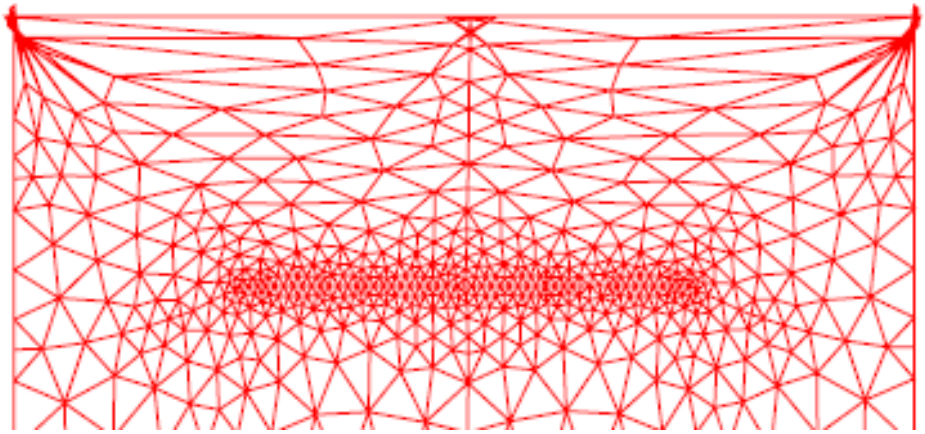

Figure 19. Effect of Co-Locating Ghost Points with Their Associated Surface Points

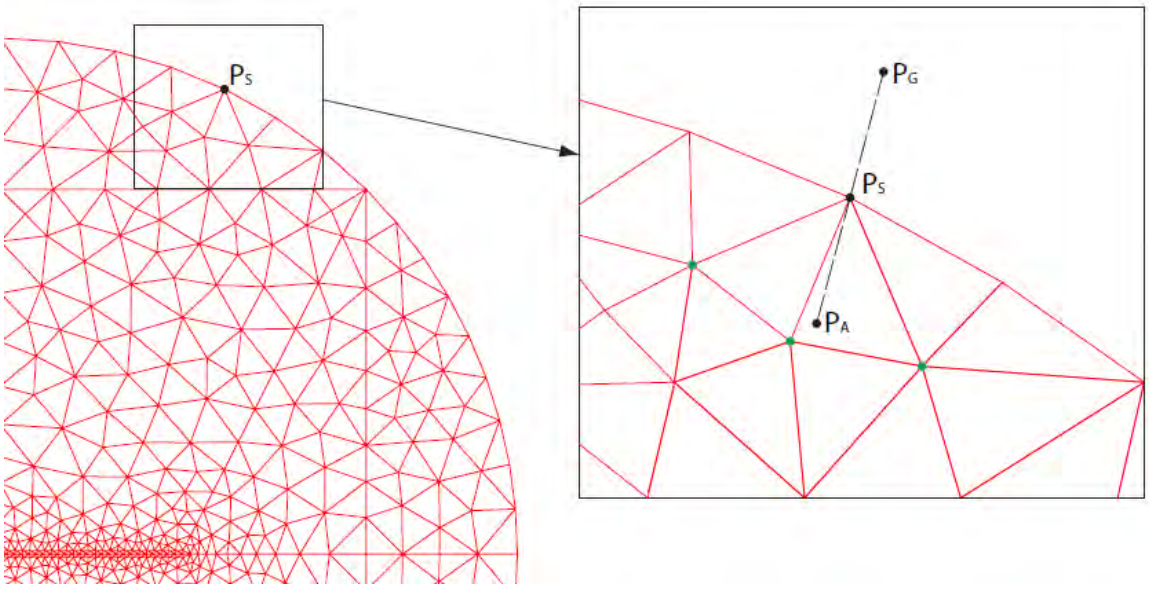

Figure 20. Placement of a Ghost Point

The next attempt at defining the coordinates of the ghost point involved calculating the average location of all interior points (P_A) connected to a surface point (P_S) and then extending the line connecting P_A and P_S. Once the line connecting P_A and P_S was extended, a ghost point (P_G) was then placed on this extended line at a distance equal to the distance between P_A and P_S.

To graphically illustrate the placement of the ghost points, consider the surface point shown as P_S in Fig. 20. P_S has 3 interior nodes to which it is connected; these points are shown in green in Fig. 20. The average interior point (P_A) is placed at the average coordinates of the three attached points. A line is then formed between P_A and P_S using the point slope equation of a line $\left(y - y_1 = m\left(x - x_1\right)\right)$ and a ghost point, P_G, is placed on that same line at a distance outside the boundary equal to the distance between P_A and P_S.

4.1.1 Reflection vs. Extension

Placing the ghost points using the extension technique described above gave reasonable results, but in order to most efficiently place the ghost points, the methodology used for the ghost point placement was explored further. As described previously, the first viable method for placing the ghost points involved extending a line connecting P_A to P_S and placing P_G on that line at a distance equal to the distance between P_A and P_S. Another possible technique would be to use a direct reflection of P_A about the boundary surface at P_S. The contrast in the placement of the ghost point using these two techniques is illustrated in Fig. 21.

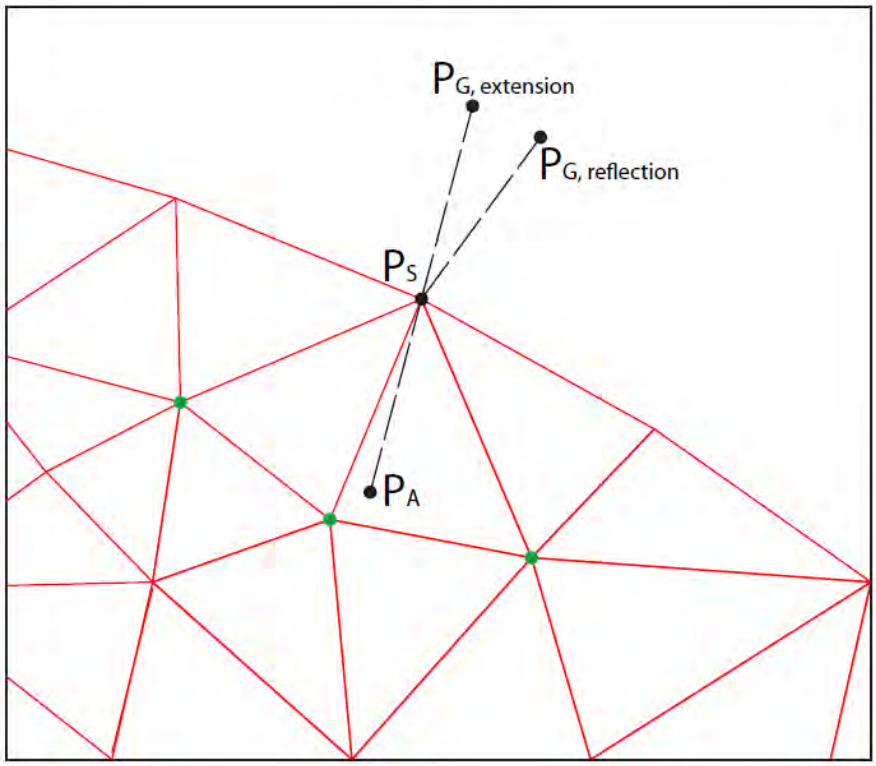

Figure 21. Ghost Point Placement: Reflection vs. Extension

Intuitively, from examining Figs. 20 and 21 it appears that placing the ghost point using reflection would tend to be a better convention. Looking at the mesh in these figures, which has a circular outer boundary, it can be seen that P_A is at a location which is generally clockwise from P_S. However, placing the ghost point using extension will put the ghost point at a position which is slightly counterclockwise to P_S, thus diminishing some of the force that is driving the surface point in the desired direction (toward the average of the connected interior points). By using reflection to place the ghost point, the ghost point complements the interior points in driving the surface point to a new location rather than counterbalancing them.

4.1.2 Reflection Methodology

If reflection is to be used, a matrix of transformation is used to place the ghost point by mapping P_A onto a reflected position. This transformation, $T: \square^2 \rightarrow \square^2$, has the form:

$$T(\vec{x}) = [R]\vec{x} \tag{4.3}$$

In Eq. (4.3), $[R]$ is a standard matrix of linear transformation and $\vec{x}$ is the coordinate vector of the point that is being mapped to a new position. In order to calculate the standard matrix of reflection, one must construct the tangent vector, $\vec{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$, about which the point will be reflected. This tangent vector will represent a line tangent to the boundary surface at the point P_S. For a discretized boundary surface, the tangent can be approximated as $P_{s+} - P_{s-}$ where P_{s-}

is the point on the surface before P_S and P_{s+} is the point on the surface after P_S. Using the tangent vector of the line tangent to the surface at P_S, the standard matrix of reflection can be found as Eq. (4.4), and once the matrix of reflection is known, the ghost point can be calculated using Eq. (4.5).

$$[R] = \frac{1}{d_1^2 + d_2^2}\begin{bmatrix} d_1^2 - d_2^2 & 2d_1^2 d_2^2 \\ 2d_1^2 d_2^2 & -d_1^2 + d_2^2 \end{bmatrix} \tag{4.4}$$

$$P_G = \begin{bmatrix} P_{Gx} \\ P_{Gy} \end{bmatrix} = [R] P_A \tag{4.5}$$

As expected, using the reflection technique to place the ghost points did improve convergence. However, the effect was not drastic. Figure 22 shows a comparison that illustrates the difference in using the two techniques. It can be seen that after 1,000 iterations the mesh that is generated using the reflection technique (shown in magenta) is "leading" the mesh that is generated using the extension technique (shown in blue) as the two meshes advance toward a converged solution. The effect is most obvious by observing lines coming off the ends and the center of the flat plate, which are seeking a state such that they are normal to both the inner surface (the flat plate) and the outer boundary surface.

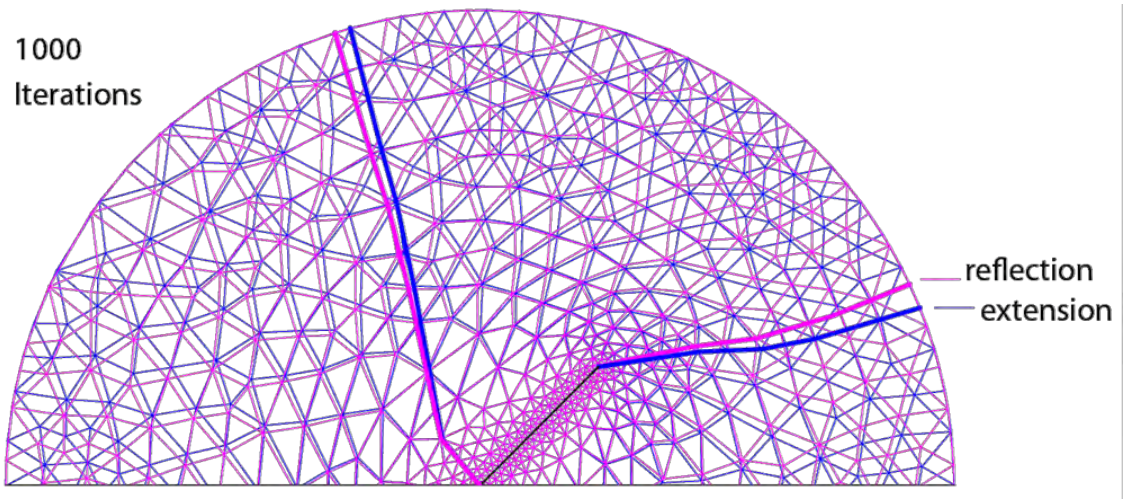

Figure 22. Convergence Comparison for a Mesh Using Extension and Reflection to Place the Ghost Nodes

4.2 ANALYTICALLY DEFINED BOUNDARIES

4.2.1 Flat Boundaries

The mesh smoothing algorithm that performs the mesh movement is a point iterative scheme where the coefficients in the Winslow equations, α, β, and γ, must be recomputed at each iteration. Each time an iteration is performed, all of the nodes will move to a point that the smoothing algorithm considers the optimal placement for that iteration. For interior points, this new position will be the starting place for the next iteration. However, for boundary points there is an additional step. At each iteration, each boundary point that has moved must be projected back to the surface that defines the original boundary. If this is not done, the boundary will begin

to wander away from its original position to a position that better satisfies the elliptic smoothing equations and will no longer conform to the true boundary.

The first case that was used to test the methodology for manipulating surface points using the Winslow equations was a flat plate surrounded by an unstructured mesh bounded by flat (constant in x or constant in y) boundary surfaces. This was the simplest analytical case, where the boundaries were defined as being linear and constant in either the x or the y direction. A mesh on which this would apply can be seen in Fig. 23. Recall that the Winslow equations are a set of two equations (for each node) that are solved for the x and y coordinates. Therefore, in the case where the boundary is flat (horizontal or vertical), only one of the equations will require a solution. For example, on the top boundary of Fig. 23, the y-coordinate will not be changing, so the second Winslow equation will not require a solution.

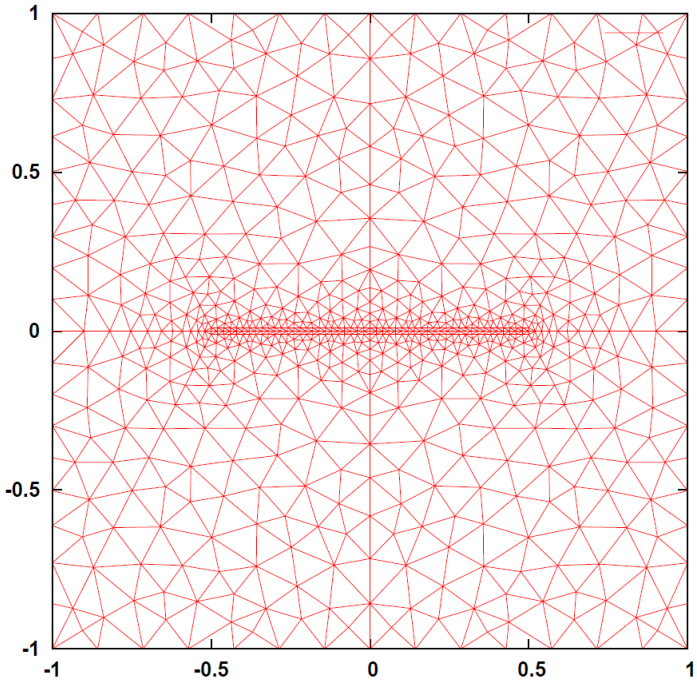

Figure 23. Flat Plate Bounded by Flat Boundary Surfaces

The mesh shown in Fig. 23 gave a good initial testbed for looking at the effects of a moving boundary, but it is of limited practical value because it has such a rigorous definition of what shape the outer boundaries must be (i.e., they must be constant in x or constant in y). For the cases with the flat outer boundaries, the implementation of the Winslow equations explicitly took this into account. The equations were implemented by only solving the equation for the coordinate that was moving. The nodes on the top boundary of Fig. 23, for example, were not allowed to move in the y direction, so the Winslow equation for movement in y did not even need to be solved. In general, this is not the case, and thus the Winslow equations for both x and y must be solved and the surface points must then be manipulated so that they conform to the original surface definition.

4.2.2 Boundaries Defined by Functions

A treatment for boundaries defined using an analytic function was also briefly explored. It has been the author's experience that the most common analytically defined outer boundary used in

two-dimensional external-flow CFD is a circle, so this is the boundary function that was used to explore this methodology. A circular boundary for a region where the center is at the origin can be defined by the expression $x^2 + y^2 = r^2$ where x and y are the coordinates of a boundary point and r is the radius of the outer boundary. Once Winslow smoothing has been performed, x and y will be modified, and then $x^2 + y^2 \neq r^2$. The boundary point must then be moved to the closest point which is on the original boundary surface definition. It can be shown that for a point, $p = (x_p, y_p)$, the closest point on a circle with radius r is the point of intersection between the circle and a ray originating from the origin and passing through point p. This logic can be used to keep the boundary points on the defined boundary but is still not ideal because it is not general enough to be used on arbitrary boundaries.

4.3 EXPLICITLY DEFINED BOUNDARIES

Although meshes used in CFD analysis sometimes have boundaries defined by analytical functions as described in Section 6.2, it is more common for boundaries to be defined explicitly by a discrete shape, which can be defined in a geometry file. The nature of the geometry file will vary, but for two-dimensional analysis the geometry file will generally take the form of a segment file where the shape is defined by discrete line segments. Any arbitrary level of accuracy in capturing the boundary shape or curvature can be achieved by increasing the number (and shortening the lengths) of the line segments in the geometry file. In three-dimensional analysis, the file will generally take the form of a tessellated surface file such as a VRML or STL file. Because boundary surfaces are most commonly defined in this manner, it is necessary that a floating-node boundary on a mesh which is to be smoothed is able to adhere to a boundary explicitly defined in this way.

Projection onto an explicitly defined boundary is achieved by utilizing a C++ “geometry” class that was developed by Dr. Steve Karman at The University of Tennessee at Chattanooga (UTC). Using this class, the boundary is defined in a geometry file, which is comprised of a user-defined number of line segments on the surface. The number of segments will vary depending on the desired tolerance of the projection onto the surface. The geometry file on which the geometry surface is based can be easily generated using Gridgen by exporting a curve (or number of curves) under the INPUT/OUTPUT commands in the Gridgen interface (Ref. 13).

Initially, the floating boundary-node logic was examined using a one-unit-length flat plate in an unstructured region surrounded by a circular outer boundary with a radius of $\sqrt{2}$. If the flat plate, which is relatively close to the outer boundary, is allowed to float within the region while the outer boundary is held static, it is difficult for the mesh to maintain high quality. This is because the connectivity between the inner and outer boundaries does not change, so the unstructured elements must stretch to try to accommodate the new orientation of the inner boundary (the flat plate). Eventually the stretching will result in unacceptable levels of skewness in the mesh and even negative volumes as mesh lines begin to cross. This is illustrated by Fig. 24, which shows the mesh as the flat plate is rotated to 45 and 90 deg while the nodes on the boundaries are held static.

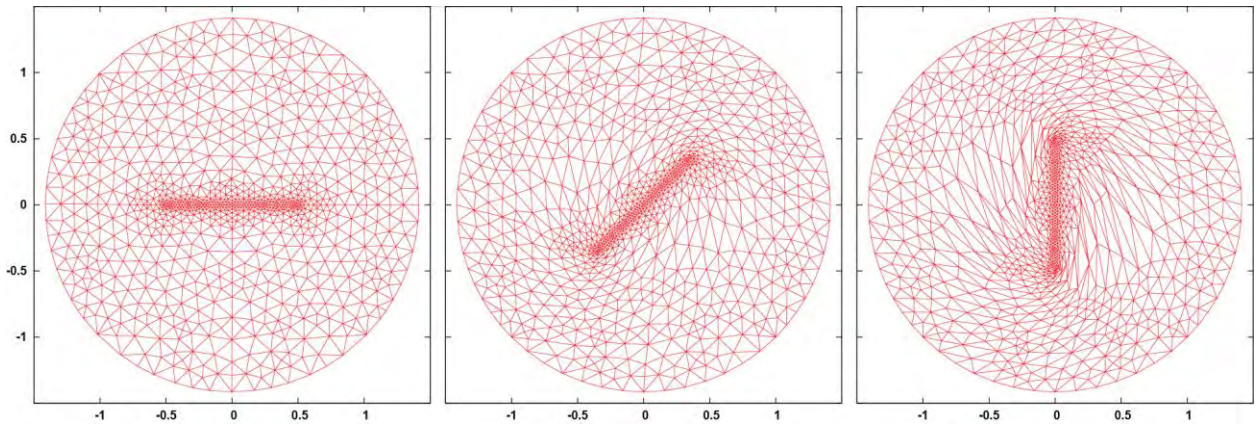

Figure 24. Flat Plate Rotated 45 and 90 deg While Boundary Nodes are Held Static.

When the outer boundary nodes are allowed to float and the smoothing logic for boundaries is employed, the results for the rotated flat plate are shown in Fig. 25. An interesting comparison can be made between the rotated mesh and the mesh in its original smoothed position (i.e., when the mesh has been smoothed but the flat plate has not been rotated). The mesh, including the outer boundaries, looks the same except that they are rotated 45 and 90 deg.

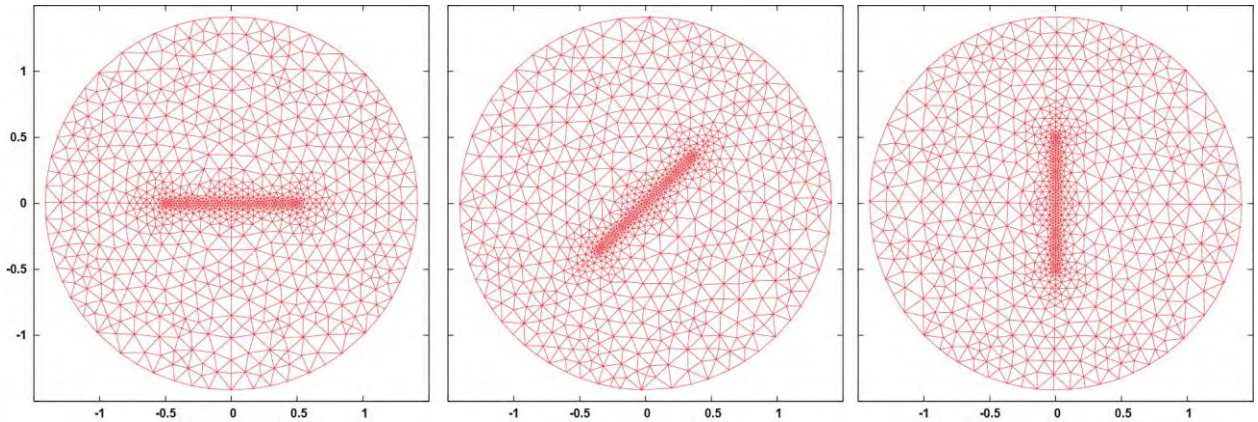

Figure 25. Flat Plate Rotated 45 and 90 deg While Smoothing is Performed on Outer Boundary Nodes

4.4 FLOATING POINTS ON MULTIPLE BOUNDARIES

The Winslow elliptic smoothing code was extended to be able to handle moving points on multiple boundaries. To test this capability, a mesh was generated that had a structure similar to the flat-plate mesh, but instead of a flat plate the inner surface was a NACA0012 airfoil shown in Fig. 26.

When dealing with a shape such as an airfoil, which has two portions of the surface that are relatively close together, it is necessary to split the surface into an upper and lower portion and limit the surface-point projection to the original surface. Otherwise (e.g., at the sharp tail of an airfoil) a node from the upper surface can easily be projected onto the bottom surface if the node is moved between the two surfaces when smoothing is performed. This undesirable effect is illustrated in Fig. 27.

A separate boundary condition was added to the code to handle boundaries that were comprised of multiple sections. The boundary condition was implemented in such a way that when the mesh was smoothed, the endpoints remained in their original position while the remaining points on the boundary were allowed to float. This was different from the previous implementation where all surface nodes, even the nodes at endpoints, were allowed to float. When this new boundary condition was implemented, the mesh surrounding the interior (airfoil) surface acted as expected. These results can be seen below in Fig. 28.

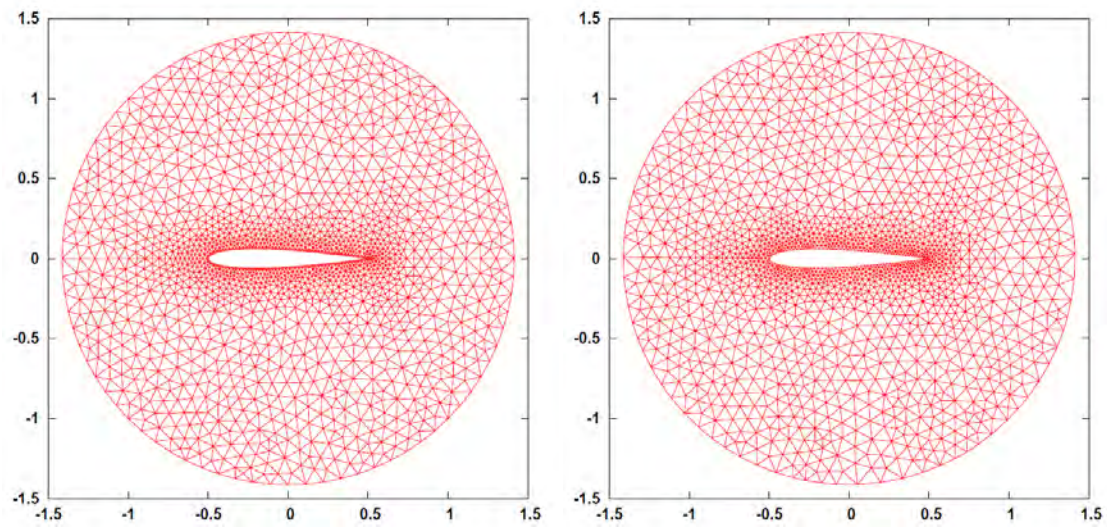

Figure 26. Airfoil Mesh Before and After Smoothing

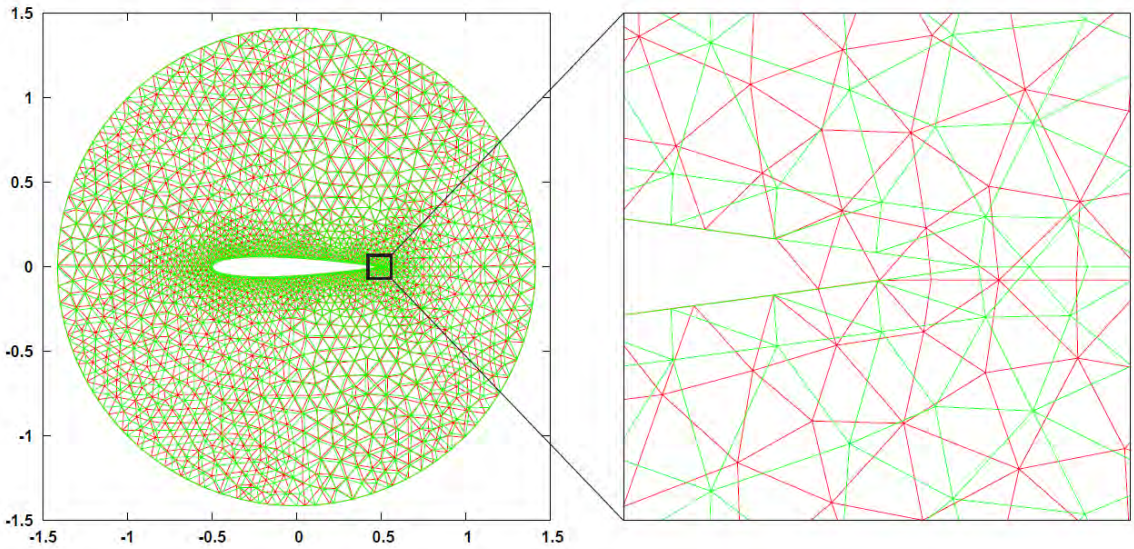

Figure 27. Effect of Projecting Onto the Wrong Surface of a Sharp Airfoil Tail

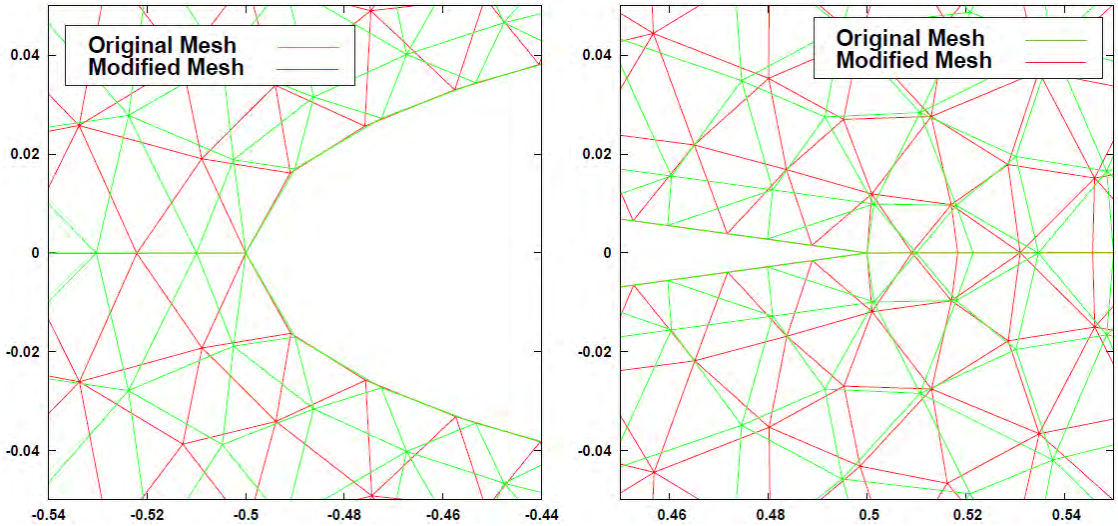

Figure 28. Close-Up of Mesh Near Interior Surface at Nose and Tail of Airfoil

In Figure 28, the airfoil surface was defined as two distinct surfaces (a lower surface and an upper surface) and the surface mesh was confined to its original surface. This eliminated the possibility of a node from the upper surface being projected onto the lower surface if the smoothing equations placed the node between the two original surface locations.

As seen in Fig. 28, the modified mesh around the airfoil surface is comprised of what would normally be considered very “good” triangles in that they are close to equilateral. However, note from Fig. 28 that in order to equalize the angles in the surface triangles, the mesh points were driven away from the airfoil surface. This might by a good mesh for an inviscid flow, but if it is desired to capture the viscous effects of a flow within a viscous boundary layer, this will have a negative impact on capturing the correct characteristics of the flow in this region. To alleviate this effect, the smoothing logic must be modified such that the computational space (which currently utilizes equal angles) can be utilized to try to better retain the characteristics of the original mesh. This is discussed in detail in Section 7.

4.5 MULTIPLE-ELEMENT AIRFOIL

4.5.1 Airfoil Rotation

The final case that was examined to explore the effects of applying the Winslow equation to boundaries was to apply Winslow elliptic smoothing to a multi-element airfoil. The airfoil that was employed for this purpose was the 30P30N multi-element high-lift airfoil. The 30P30N is a three-element airfoil consisting of a central airfoil section with a flap section in the rear and a slat section in the front. The mesh that was used was a relatively dense mesh with 23,012 mesh points and 44,437 triangular elements. The original configuration of the airfoil was with the flap and slat extended (the high-lift configuration); the entire mesh can be seen in Fig. 29 and a close-up view near the airfoil surface can be seen in Fig. 30.

The 30P30N airfoil was pitched 30 deg counterclockwise, and the surrounding mesh was smoothed using the Winslow equations to adapt the flowfield mesh to the new airfoil surface position. The mesh was smoothed using both a static outer boundary and a floating-node outer boundary, which can be seen on the left and right sides of Fig. 31, respectively. Initially, because of the dense mesh and the abundance of levels of connectivity between the inner and

outer surfaces, it was unclear whether the outer boundary would be greatly affected by the treatment of the outer boundary nodes (static vs. floating). After the cases were examined, it was determined that, in fact, the outer boundary was significantly affected. As shown in Fig. 31, when a floating-node boundary was employed, the nodes on the outer boundary floated in a counterclockwise direction until the final mesh closely resembled the original mesh, but rotated 30 deg. Four representative outer boundary mesh points are marked in black in Fig. 31 so that the rotation can be easily discerned.

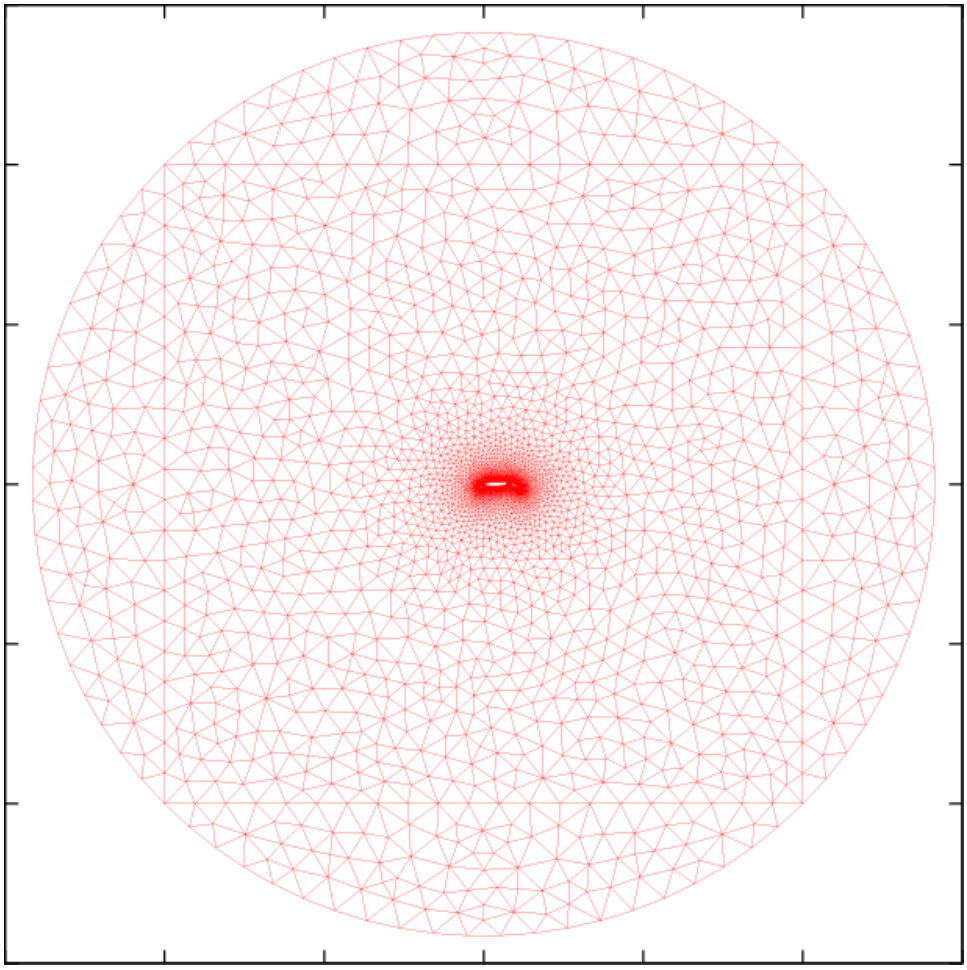

Figure 29. 30P30N Airfoil and Mesh

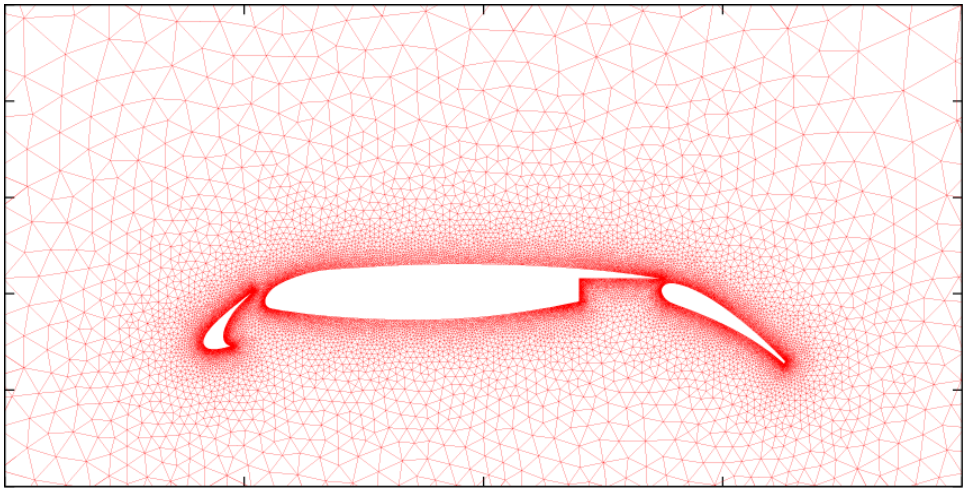

Figure 30. Close-Up of 30P30N Airfoil

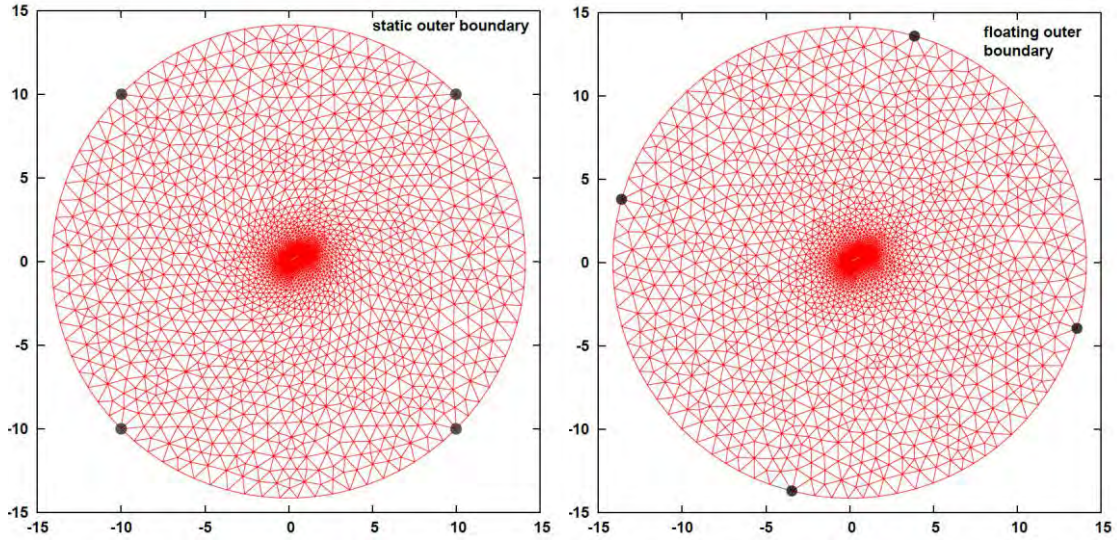

Figure 31. 30P30N Mesh, Without (Left) and With (Right) Floating-Node Outer Boundary

A close-up of the mesh near the airfoil (before and after the 30-deg pitch) is seen in Fig. 32. In order to test the quality of the mesh, a CFD flow solution was generated using an in-house 2D Euler solver for the airfoil at both 0 and 30 deg angle of attack using a freestream Mach number of 0.95, and the results are shown in Figs. 34 and 35. The solver did not have any problems generating a flow solution on either the zero angle of attack or the mesh that had been rotated 30 deg and then smoothed using the Winslow equations.

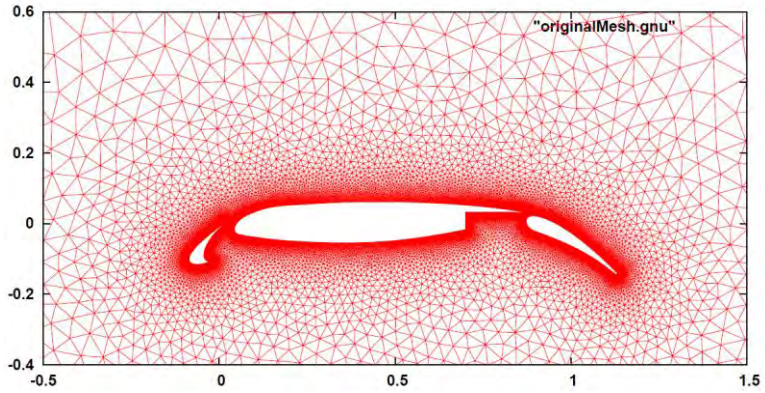

Figure 32. 30P30N Airfoil at 0 deg Angle of Attack

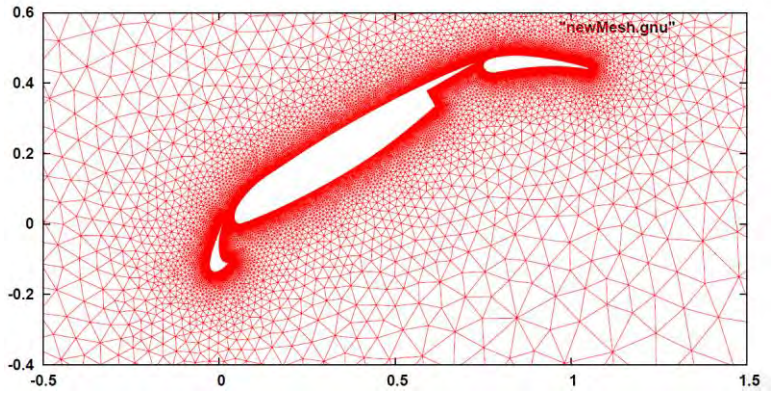

Figure 33. 30P30N Airfoil at 30 deg Angle of Attack

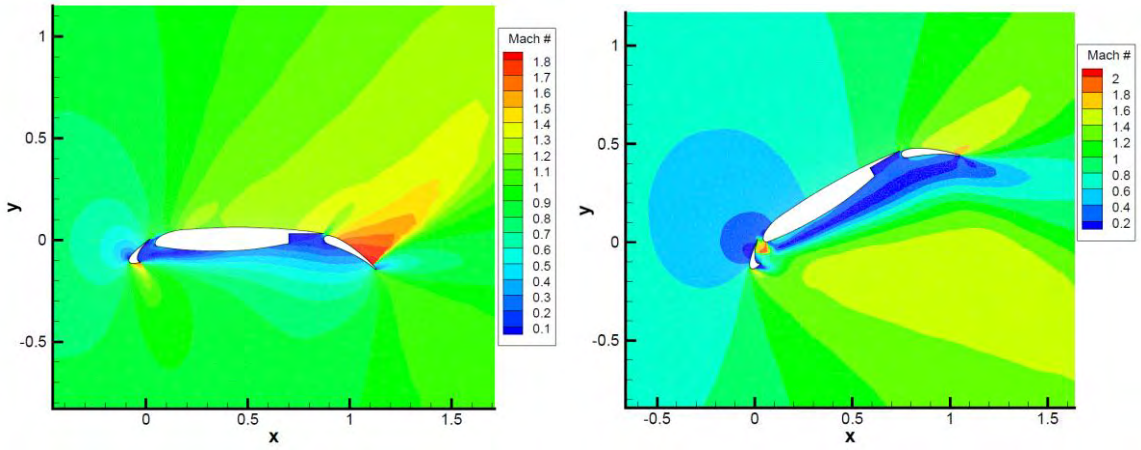

Figure 34. CFD Solution on a 30P30N Airfoil

4.5.2 Airfoil Slat Movement

Because the goal of analyzing the 30P30N airfoil was to demonstrate the power of the Winslow equations applied to a boundary, it makes sense to look at how two surfaces that are close together behave if one of the surfaces is moved relative to the other. For this, the slat at the

front of the airfoil is rotated from its original extended (high-lift) position to a retracted position. The slat is rotated over two steps, and the change in position can be seen in Fig. 35.

The mesh was smoothed at each of the two slat rotation steps. The first case that was examined was using static nodes on all of the airfoil boundaries. When this was done, the mesh between the slat and the central airfoil section became highly skewed. These results can be seen in Fig. 37. The next case that was examined involved letting the boundary points on the surface of the central airfoil section float as the slat was rotated. This gave a much better mesh near the leading edge of the central airfoil section, and these results can be seen in Fig. 38.

The results from the exploration of the 30P30N multi-element airfoil show one practical “real world” case where utilizing the ghost node technique for applying the Winslow elliptic smoothing equations to the boundaries of an unstructured mesh resulted in a significant improvement compared to the case where static-node boundaries were used and only the interior mesh was smoothed.

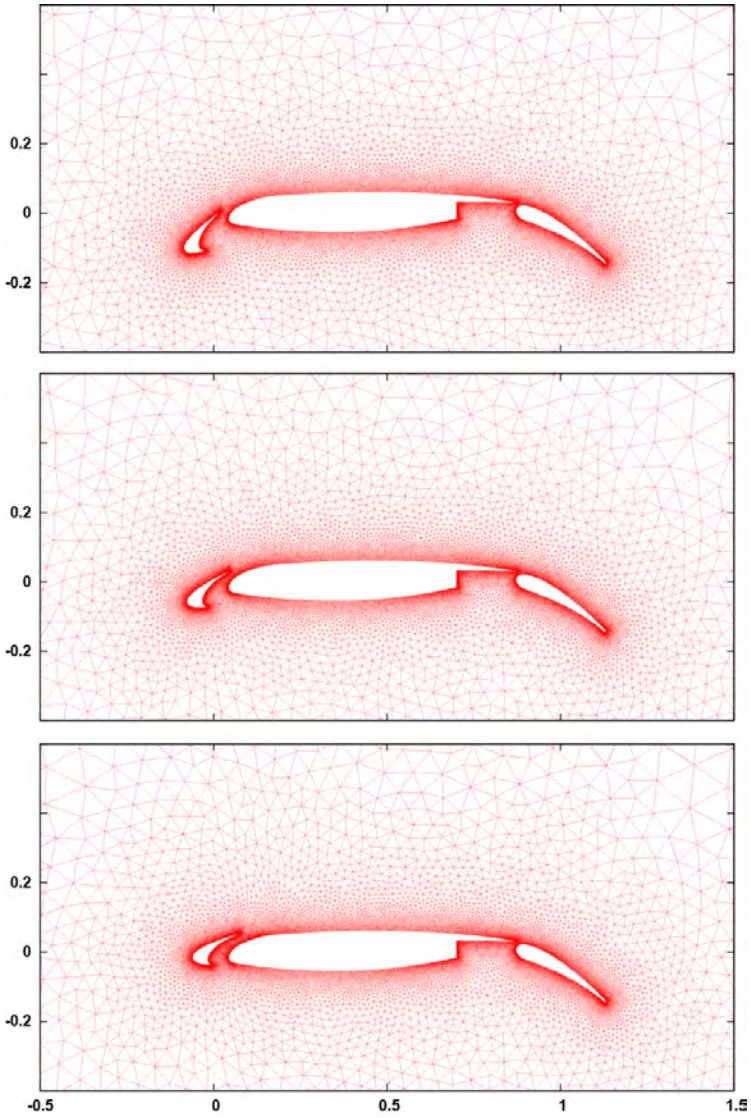

Figure 35. 30P30N Airfoil with Front (Slat) Section Moved from Extended to Retracted Position

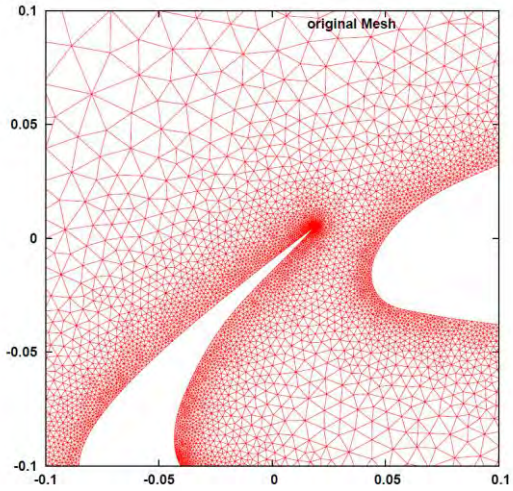

Figure 36. Original Slat Position

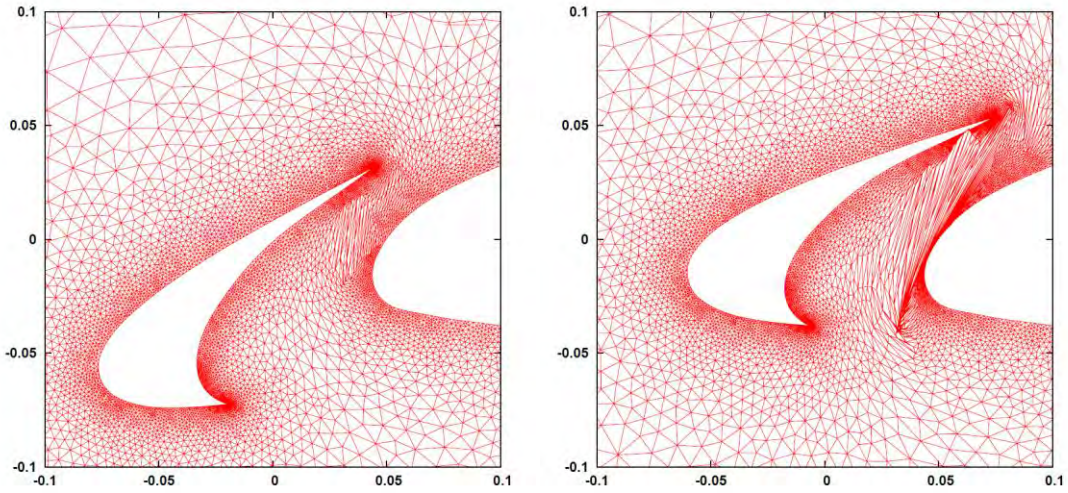

Figure 37. Slat Movement Using Static Surface Nodes

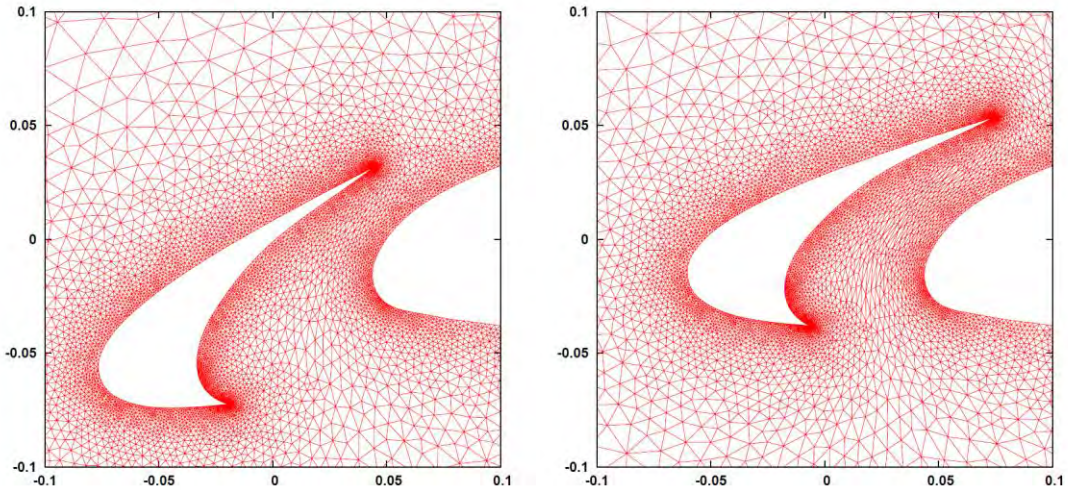

Figure 37. Slat Movement Using Floating-Node Boundary on Central Airfoil Section

5.0 ITERATIVELY ADAPTED COMPUTATIONAL SPACE

A technique was developed that allows for the smoothing of a viscous mesh with general spacing properties even if a viable viscous mesh is not originally available. The technique requires that the mesh have a connectivity structure in which each node in the viscous region has an associated surface node, as would be the case if a marching technique was used to generate the viscous region of the mesh. The mesh is smoothed in such a way that a desired viscous profile can be achieved based on an initial off-body spacing and the geometric progression of the viscous layers. The initial mesh that was used to experiment with this new technique was an inviscid (isotropic) mesh around a rough airfoil. The mesh is shown below in Fig. 39 and, near the airfoil, in Fig. 40.

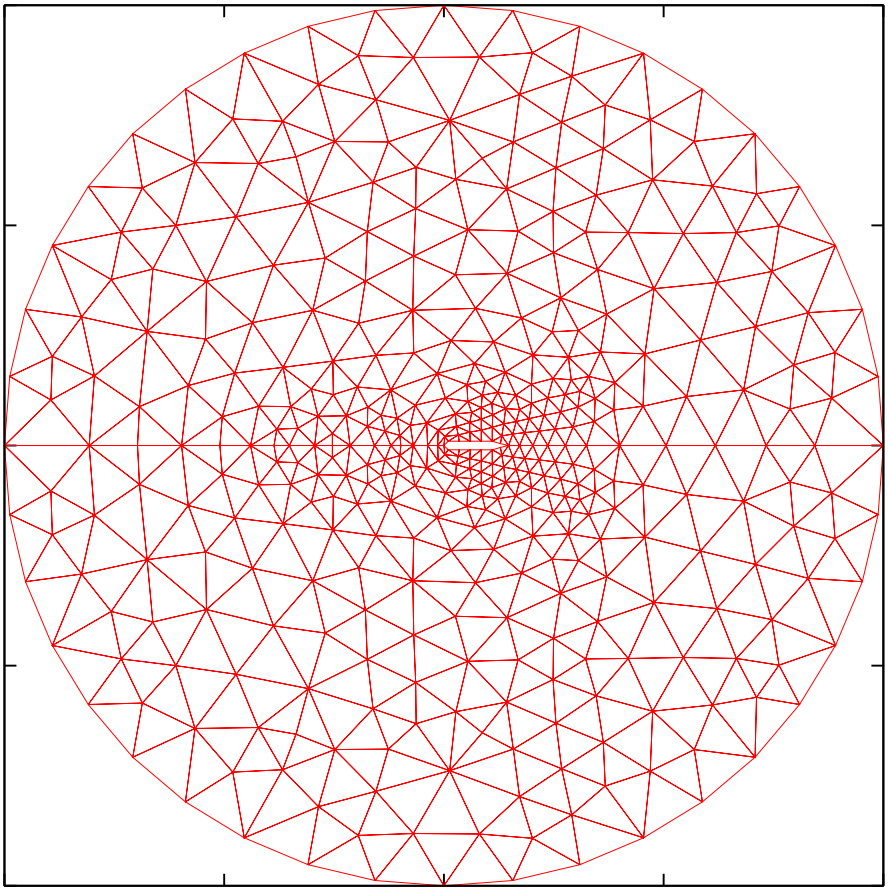

Figure 38. Full Domain of Inviscid Mesh Surrounding Rough Airfoil

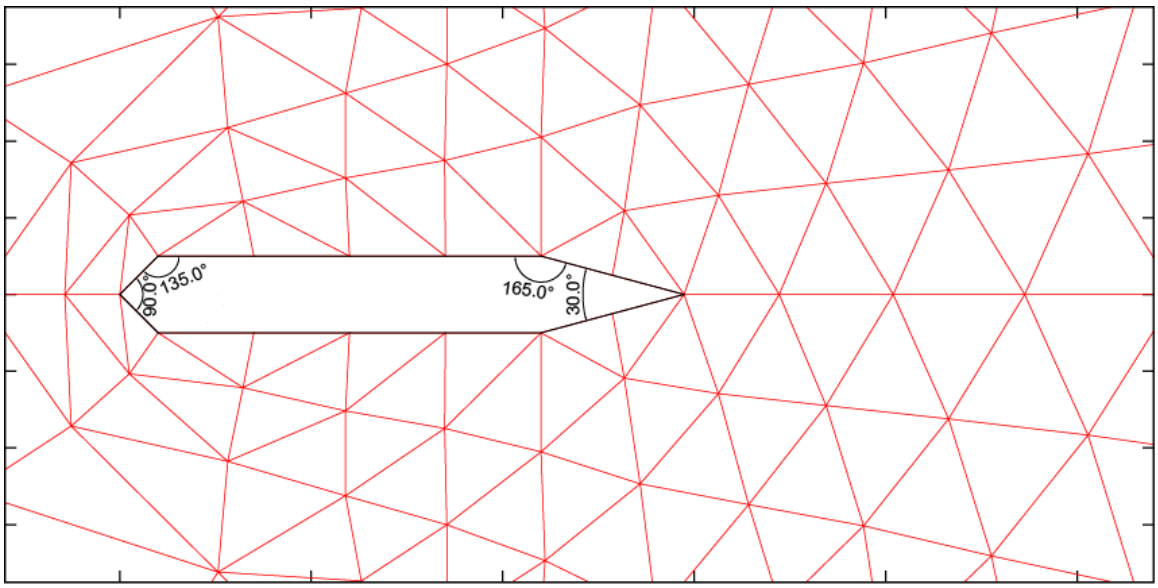

Figure 39. Inviscid Mesh Around Rough Airfoil Near Airfoil Surface

5.1 NORMAL OFFSET

Throughout the development of the iteratively adaptive technique and algorithm, the computational space that is used as the starting point on which to adapt is an equal angle, equal edge-length virtual control volume such as one of the virtual control volumes shown in Fig. 7. The initial step in the development of an iteratively adaptive viscous smoothing algorithm was to use a geometric progression factor to create the offset in computational space that would be normal (orthogonal) to the no-slip surface in physical space. The offset can be defined as the distance from the central node in computational space to the point at which the center of all the neighboring nodes is located. (A note on terminology: as mentioned in Section 1, the node referred to as the central node is the node-of-interest or the node being smoothed. It is always located at the origin in computational space, and the neighboring nodes are translated around it to accommodate the characteristics of the physical mesh.) The direction of the normal offset is determined by which of the central node's neighbors is the near-surface node (where near-surface node refers to the neighboring node either directly on the no-slop surface or on the direct path back to the surface). This near-surface node will have a given position in the list of neighbors, and that position will determine where the node is positioned in computational space.

In the case where the associated surface node is located at position ξ_1, the offset in computational space will be as shown in Fig. 41. By offsetting the computational space in this manner, the distance between the central node in computational space and the near-surface node (distance L_1) is now less than the distance between the central node and the node opposite the surface (distance L_2). This gradient in computational space causes the node in physical space to move closer to the surface as well.

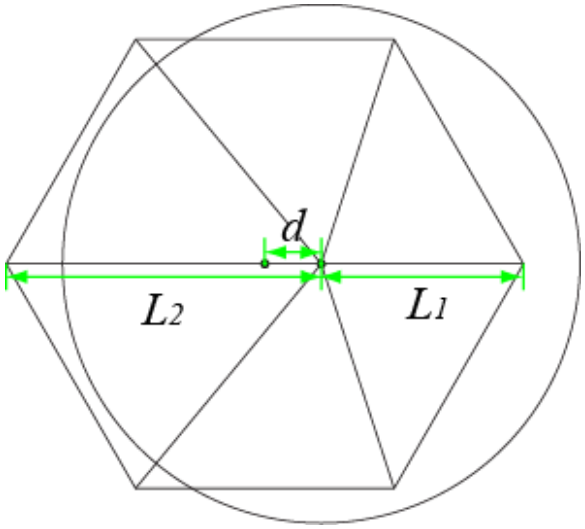

Figure 40. Computational Space Offset Normal to Viscous Surface

The magnitude of the offset (*d* in Fig. 41, where the virtual control volume is based on a unit circle; i.e., radius = 1) is based on the geometric progression factor (g) and is calculated as follows:

$$g = \frac{L_2}{L_1}$$
$$L_2 = 2 - L_1$$
$$L_1 = 1 - d$$
$$g = \frac{1+d}{1-d}$$

A little additional algebra gives the equation in terms of the offset, *d*.

$$g = \frac{1+d}{1-d}$$
$$g - dg = 1 + d$$
$$dg + d = g - 1$$

$$d = \frac{g-1}{g+1} \tag{5.1}$$

If nothing but the geometric-progression-based normal offset is used to modify the computational space control volumes, a viscous mesh can be achieved without iterative computational space adjustment, but in order to get a desired off-body spacing, the number of layers in the viscous region would have to be such that the normal distance of the viscous region (i.e., the distance from the surface to the edge of the viscous region) was precalculated to equal the sum of the progressive distances of each layer. This is a limitation that can be overcome by adding a loop into the Winslow solver that recalculates the offset “on the fly” to match off-body spacing in the first viscous layer and the progressive spacing in the following layers. Consider the mesh shown below in Fig. 42 with an initial off-body spacing of 0.04 and a

geometric progression factor of 1.5. This will give the viscous layer distances shown in Fig. 42, where *p1*, *p2* and *p3* are all associated with the surface node *p0*.

After a determined number of smoothing iterations, the normal distances in the viscous region (i.e., the distance from the viscous node to the associated surface node) will be compared to the desired distances, and if they do not match, the computational space will be adjusted accordingly. For example, if the distance *d2* is greater than 0.6, the offset will increase. This will increase the computational space gradient in the normal direction and draw *d2* closer to *d1*. When this technique is applied to the inviscid mesh shown in Fig. 39, the mesh shown in Figs. 43 and 44 results.

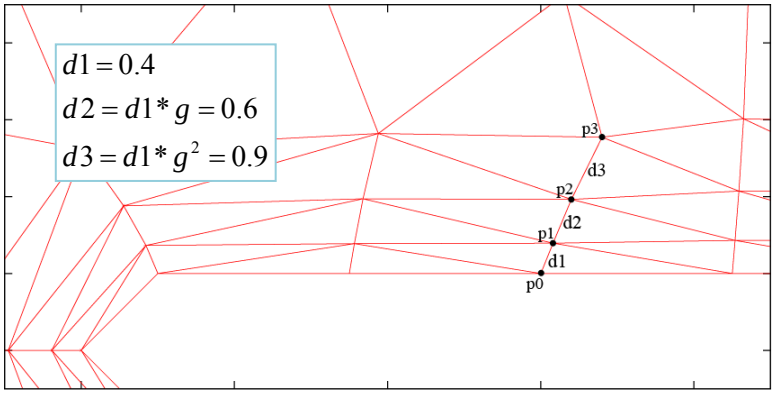

Figure 41. Mesh Smoothed Using Normal Offset

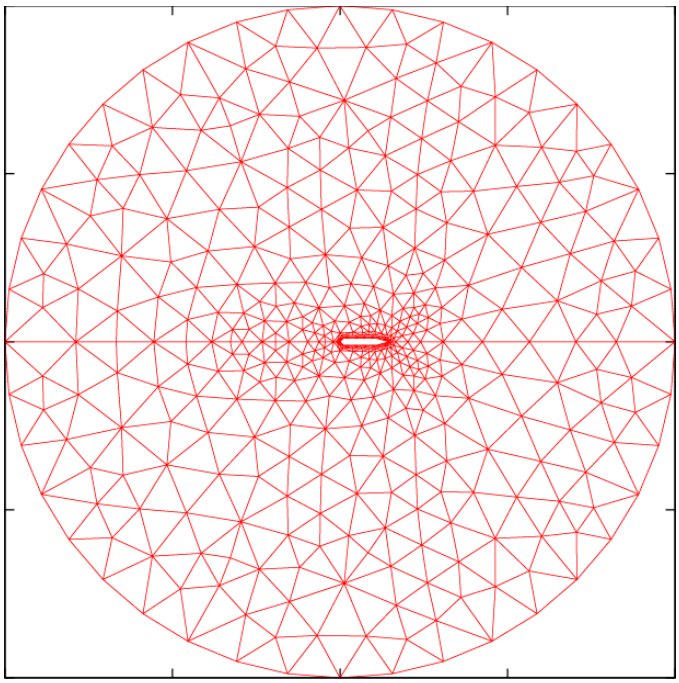

Figure 42. Rough Airfoil Smoothed Using Normal Offset Computational Space

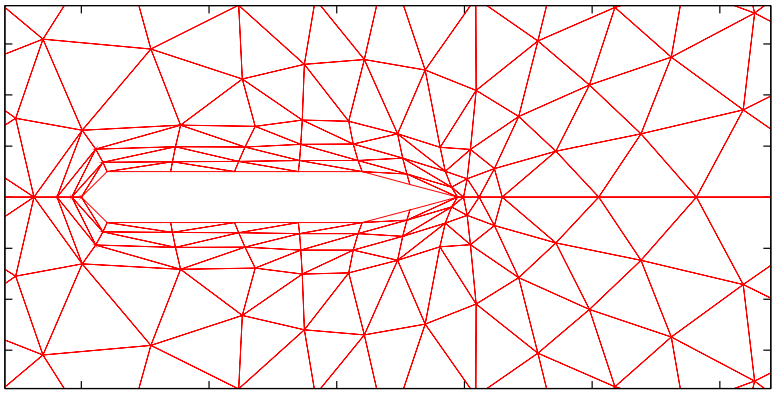

Figure 43. Close-Up of Rough Airfoil Smoothed Using Normal Offset Computational Space

5.2 NORMAL AND LATERAL OFFSET

As shown in Figs. 43 and 44, the spacing profile of the viscous layers is reasonable. However, using only the normal offset algorithm, it is not possible to manipulate the spacing in the lateral direction (parallel to the surface), so the nodes in the viscous region do not remain orthogonal to their associated surface point. This results in the situation shown in Fig. 45 where the angle, α, begins to get larger and can result in numerical instability (Ref. 15).

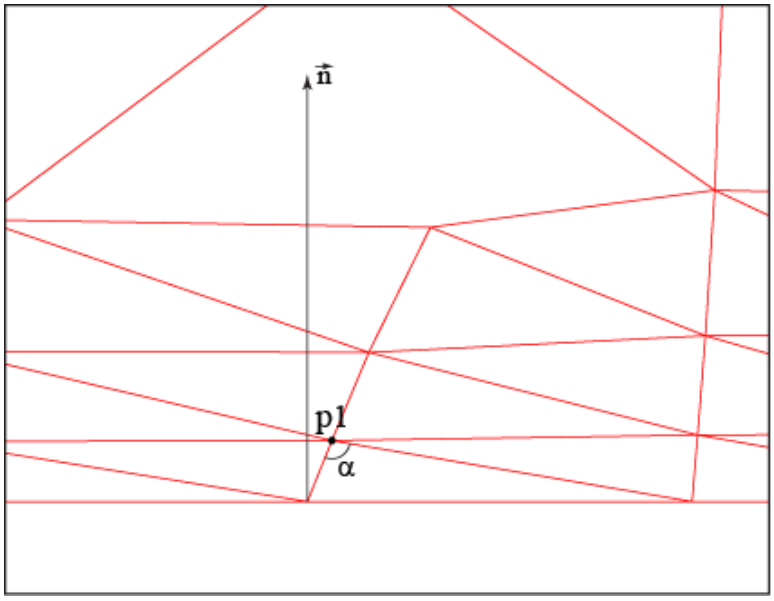

Figure 44. Normal Offset Computational Space Can Result in Large Angles

The lateral spacing issue is handled by implementing another offset in computational space where the new offset is in a direction orthogonal to the normal offset direction. The computational space will now look like Fig. 46.

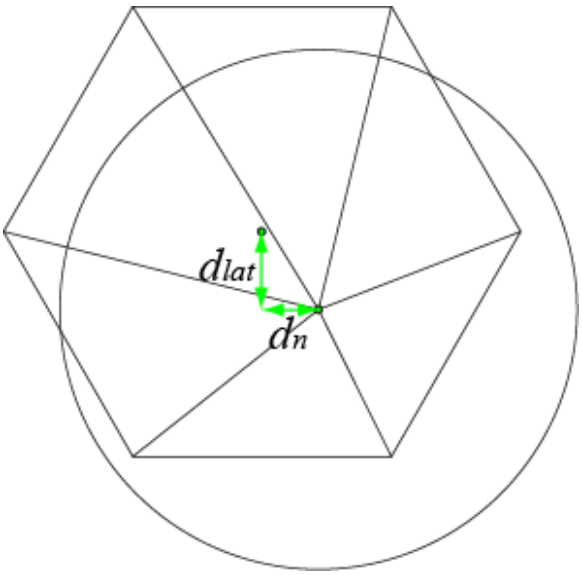

Figure 45. Computational Space with Offsets in Both the Normal and Lateral Directions

There are two possibilities for the lateral offset: normal offset direction 90 deg or normal offset direction -90 deg. The direction is based on which side of the surface normal the node is on. However, this can be handled implicitly by using the cross product [Eq. (5.2)] of the unit normal surface vector ($\vec{n}$) and the unit vector that passes through the node that is going to be smoothed ($\vec{v}$) (illustrated in Fig. 47). The cross product of two vectors is itself a vector, but because two-dimensional meshes are being utilized, the direction will simply be $\hat{k}$ or $-\hat{k}$, so the value of the cross product can be effectively treated as a scalar value with the direction captured by the sign of the scalar.

$$\vec{n} \times \vec{v} \tag{5.2}$$

The cross product defined by Eq. (5.2) is extremely well suited for the task of smoothing a viscous node in the lateral direction. It is a powerful tool because it gives information on both magnitude and direction. By using the cross product, the initial direction of the lateral offset can always be given by the normal direction 90 deg. If the cross product is negative, the direction will effectively become -90 deg, so the direction does not have to be explicitly adjusted if the node switches from one side to the other side of the surface normal. Also, as the node gets closer to the normal direction, the magnitude of the cross product, which is equal to twice the area of the triangle formed by the two vectors ($\vec{n} \times \vec{v} = 2A$, the area is shown in light blue in Fig. 47), will get smaller and smaller, so the cross product (multiplied by some relaxation factor) intrinsically makes a powerful parameter that can be used to adjust the offset and measure convergence. Because the magnitudes of the vectors in Fig. 47 and the radius of the virtual control volume have all been normalized to 1, the initial offset can be calculated as the height (*h*) of the triangle formed by the two vectors. The area of a triangle is $A = \frac{1}{2}bh$. The base (*b*) of the triangle is 1, so *h* becomes:

$$h = 2Ab = \vec{n} \times \vec{v} \tag{5.3}$$

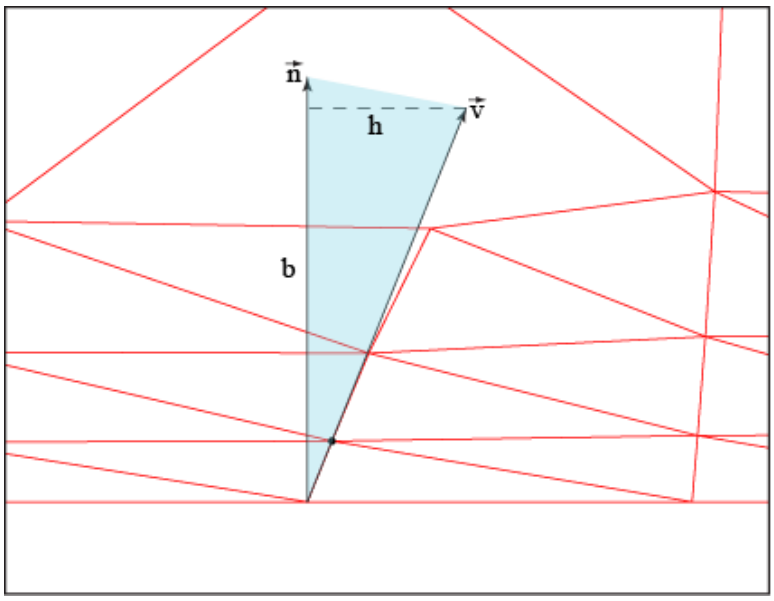

Figure 46. Area Representation of Cross Product

As the mesh is smoothed, the computational space for the viscous nodes is adjusted at given intervals (e.g., every 100 smoothing iterations). The algorithm that is used to calculate the normal and lateral offsets and adjust the computational space accordingly is described by the following:

- The initial normal offset is calculated based on the geometric progression factor (see Eq. [5.1]), which is read in as an input.

- The initial lateral offset is calculated as the distance *h* (see Eq. [5.3]) based on the cross product of the unit surface normal vector ($\vec{n}$) with the unit vector from the associated surface node to the central node ($\vec{v}$).

- The normal offset direction is calculated based on the position of the near-surface node in the array of neighboring nodes. (For the node in the first viscous layer, the near-surface node will be the associated surface node; for a node in the second layer, the near-surface node will be the associated node from layer one, etc. For a node with six neighbors, if the near surface node is first in the neighbor-array, the offset will be away from the 0-deg position; if the near-surface node is second in the neighbor-array, the offset will be away from the 60-deg position, etc.)

- The initial lateral offset direction is determined to be the normal offset direction 90 deg. (If the cross product is negative, this will effectively be -90 deg.)

- At each computational space adjustment iteration the normal offset is adjusted by 10% of its original value to target in on the offset that will give the proper off-body spacing.

- At each computational space adjustment iteration the lateral offset is adjusted by the new cross product multiplied by a relaxation parameter to target a position normal to the no-slip surface. (A relaxation factor of 0.1 has been found to give good results but might need to be decreased as the initial off-body spacing gets closer to the viscous surface.)

When the algorithm just described is implemented and applied to the rough airfoil, the results shown in Figs. 48 and 49 are achieved.

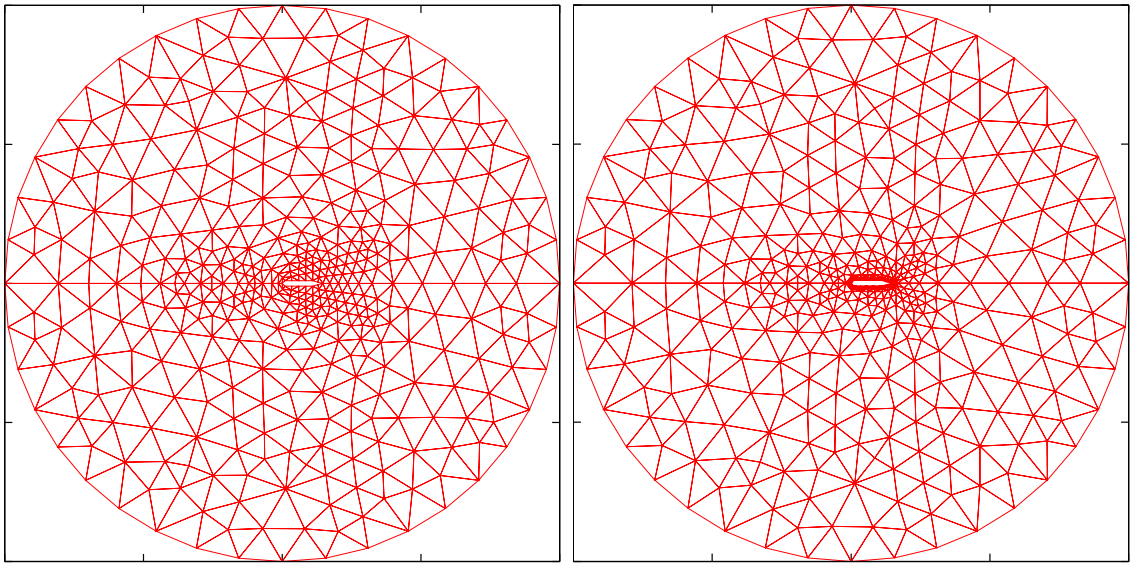

Figure 47. Rough Airfoil Mesh Smoothed Using Iteratively Adapted Computational Space in Viscous Region

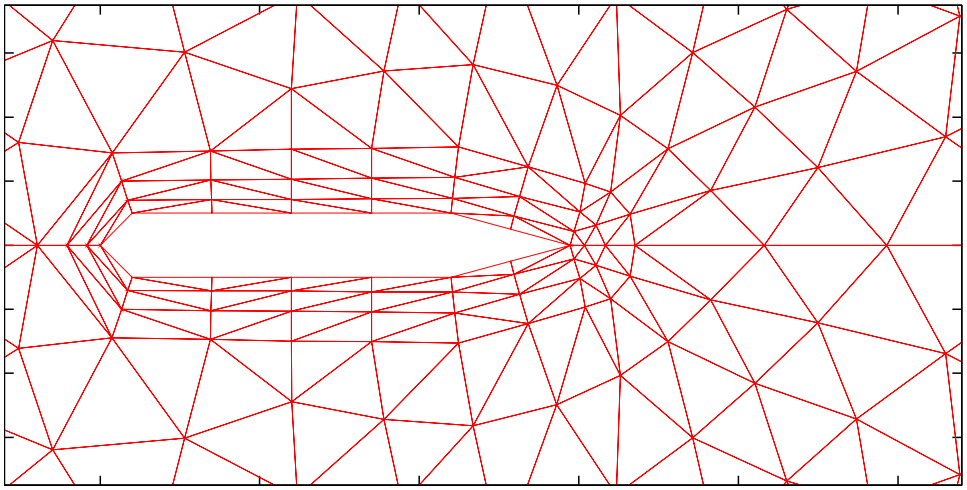

Figure 48. Close-Up of Smoothed Viscous Region Near Rough Airfoil Surface

5.3 VISCOUS NACA0012 RESULTS

To further explore the Winslow equations using iteratively adapted offset computational space, the NACA0012 airfoil was employed. The full domain surrounding the NACA0012 airfoil is shown in Fig. 50. A close-up of the original mesh is shown on the left side of Figs. 51 and 52. The mesh that was used as the starting point for the smoothing was essentially an inviscid mesh; however, the mesh was generated using extrusion off the airfoil surface so that the connectivity was amenable to treating the nodes in the viscous region as viscous layers.

The viscous region surrounding the NACA0012 airfoil was specified to have five viscous layers, and an off-body spacing and geometric progression factor were also specified. (These values

are specified as inputs in a viscous parameter file read in by the code.) Once all of the viscous region parameters were specified, the iteratively adaptive Winslow smoothing algorithm was applied to the inviscid NACA0012 mesh. The results are shown on the right side of Figs. 51 and 52. Figures 51 and 52 illustrate that the iteratively adaptive Winslow algorithm is doing an admirable job of meeting the viscous mesh requirements (i.e., mesh spacing sufficient to capture a viscous boundary layer and orthogonality of nodes in progressive layers of the viscous region).

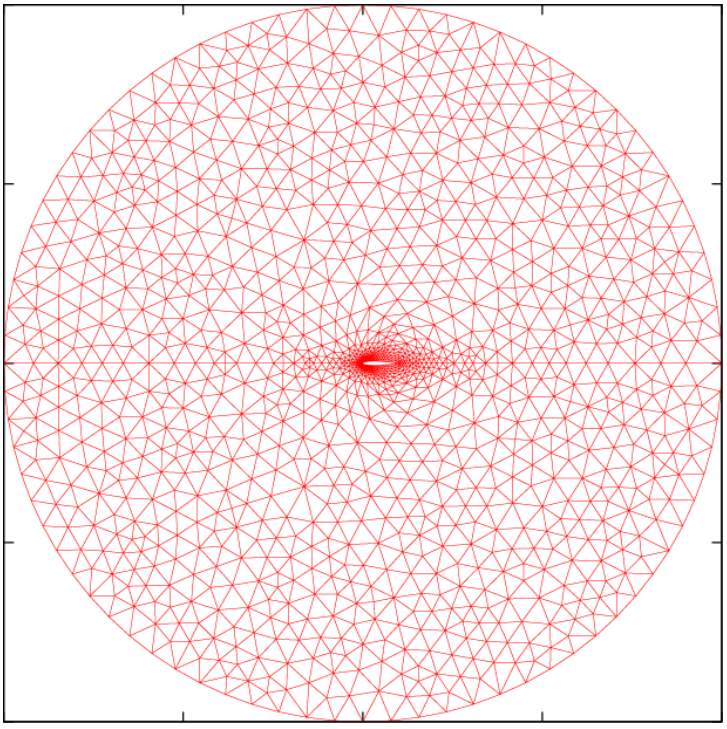

Figure 49. Inviscid Domain Surrounding NACA0012 Airfoil

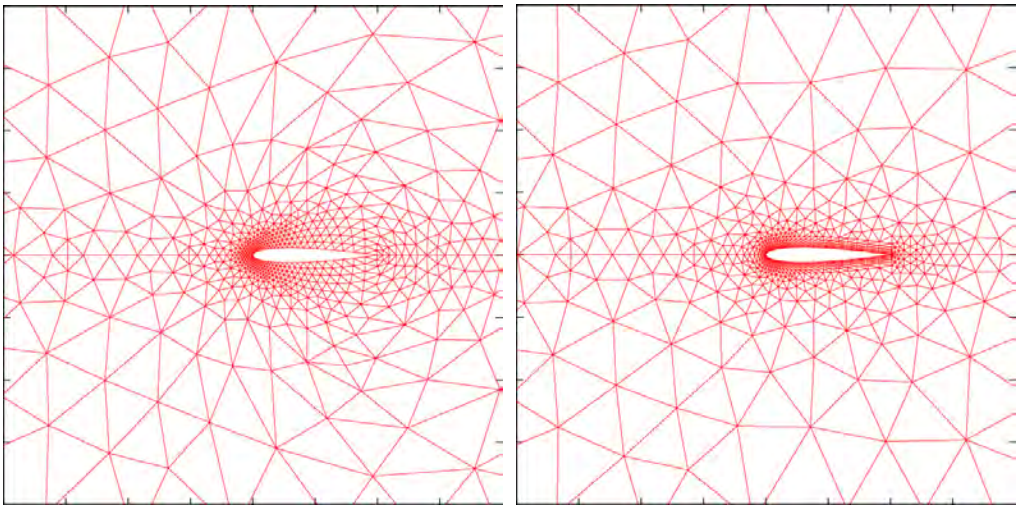

Figure 50. NACA0012 Airfoil Before and After Smoothing

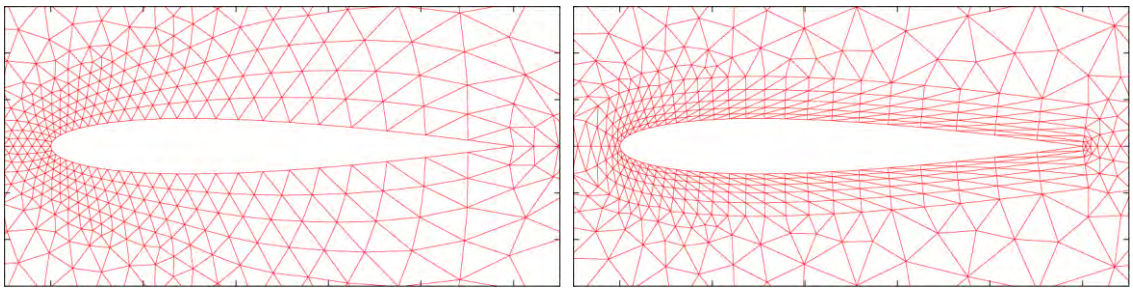

Figure 51. Close-Up of NACA0012 Airfoil Before and After Smoothing

5.3.1 Rotation and Translation

The Winslow elliptic smoothing equations using an iteratively adaptive computational space algorithm were tested on situations where an airfoil (the NACA0012) was moved around within the domain. The airfoil was rotated, translated, and then both rotated and translated simultaneously. In any of the cases, the mesh can start out as viscous, inviscid, or even an unviable mesh with grid crossing and negative volumes. The Winslow equations will cause the mesh to conform to the new position and orientation of the airfoil, and the offset computational space will maintain the viscous layer regardless of orientation. The results from the mesh movements can be seen below in Figs. 53 through 55.

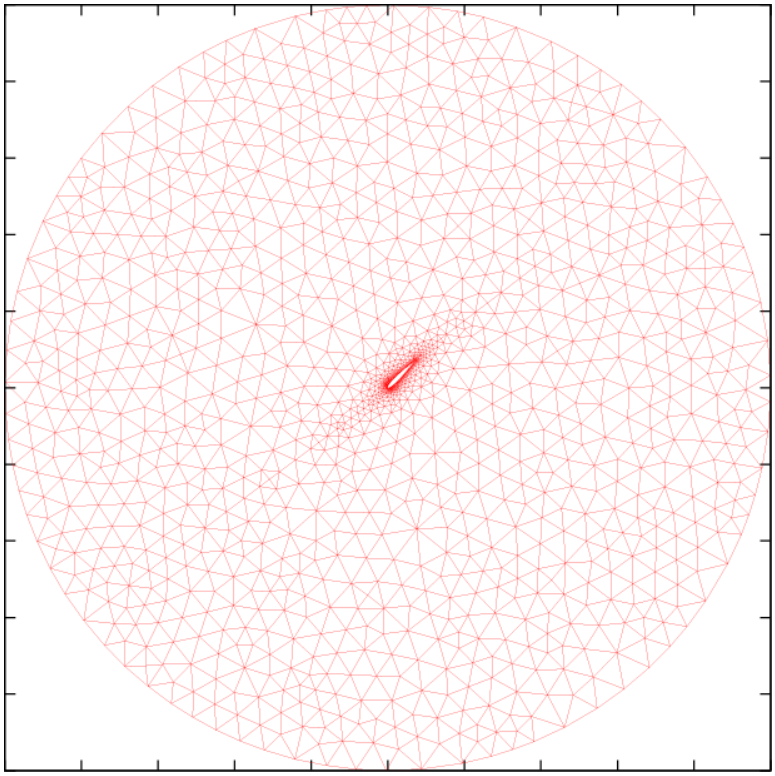

Figure 52. NACA0012 Rotation Using Winslow Iteratively Adaptive Computational Space Algorithm

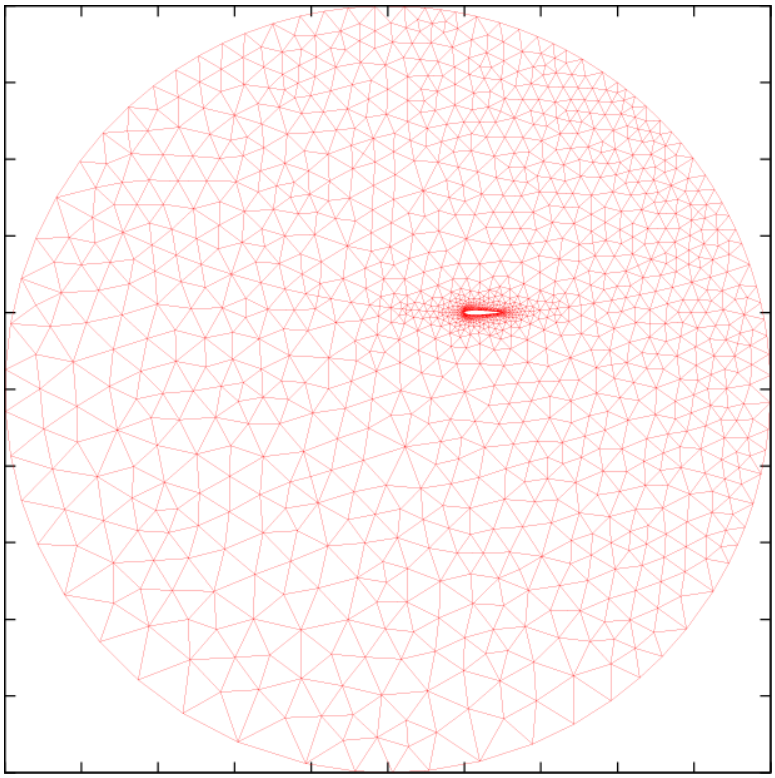

Figure 53. NACA0012 Translation Using Winslow Iteratively Adaptive Computational Space Algorithm

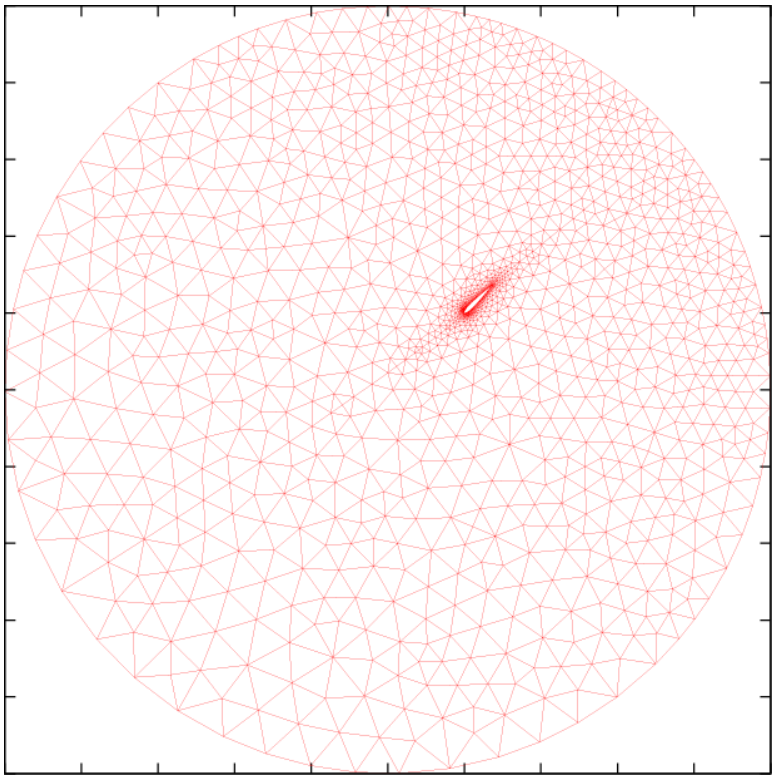

Figure 54. NACA0012 Rotation and Translation Using Winslow Iteratively Adaptive Computational Space Algorithm

5.3.2 Airfoil Shape Modification

The next area to be explored was how the equations behaved on a deforming airfoil surface. This issue has many practical applications to parametric design where it may be required that many designs be examined, but manually generating meshes for each design would be prohibitively expensive.

In many cases, the exact camber of an airfoil (for example) will not be known *a priori*. A design engineer might want to try many different incremental designs. However, the requirement that a mesh be generated for each design could potentially be significantly detrimental to the design process. By employing the Winslow equations using iteratively adapted offset computational spaces, no additional direct mesh generation would be required. The surface can be significantly deformed from its original shape and the smoothing equations will conform the surrounding mesh to the new shape and maintain the original off-body spacing and desired geometric progression. An example of this is shown in Fig. 56.

Note that even though significant grid crossing occurs when the airfoil shape is modified (second illustration in Fig. 56) the iteratively adaptive Winslow algorithm is still able to generate a viscous grid with the desired characteristics.

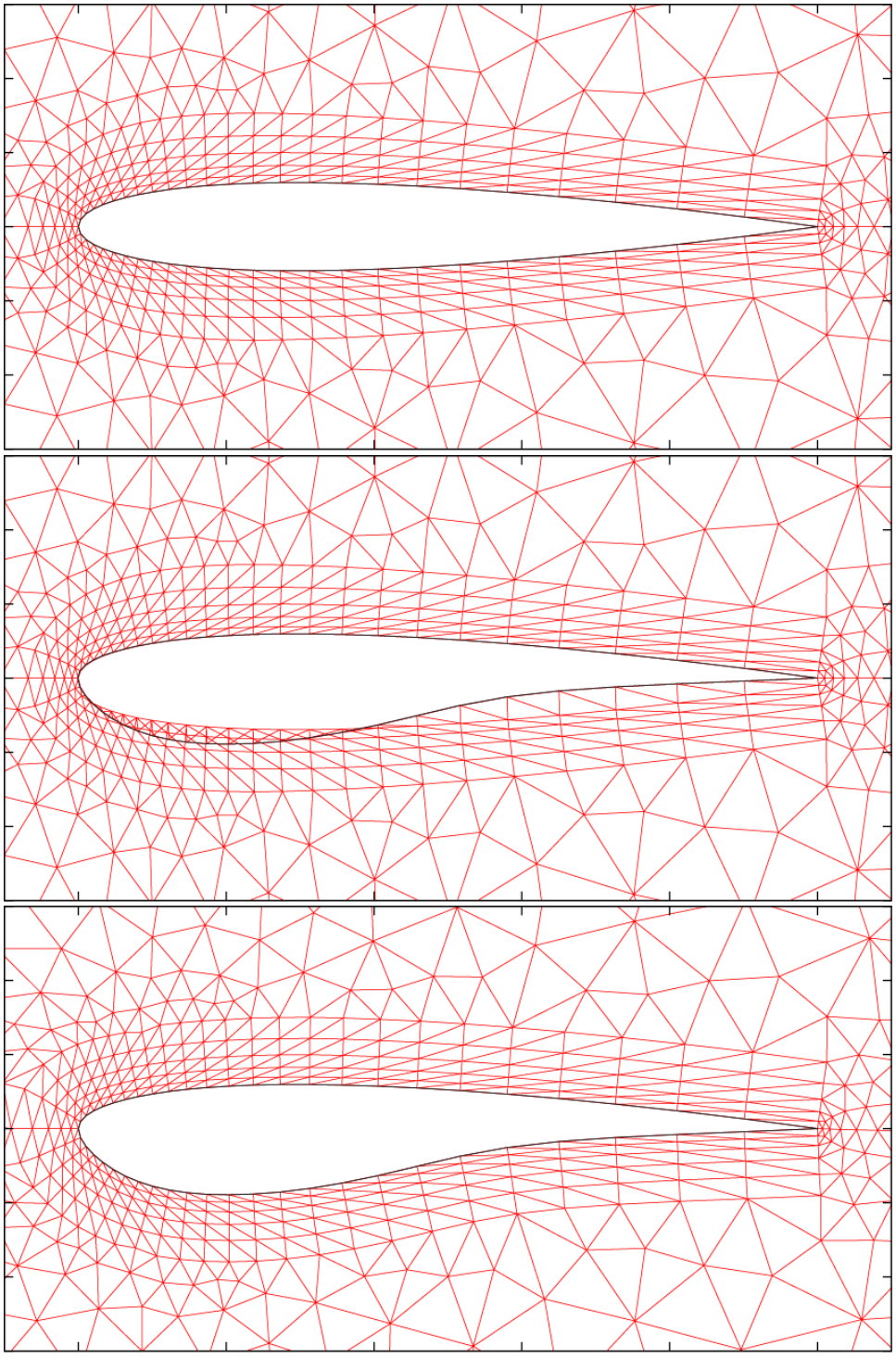

Figure 55. Viscous Mesh Conforming to a Deforming Surface

5.3.3 Change in Reynolds Number

Another powerful application of the iteratively adaptive Winslow smoothing algorithm would be in a case where the required off-body spacing was changing. This is something that is virtually guaranteed to be an issue in any kind of testing environment because it will always be

necessary to explore properties at different Mach numbers and atmospheric conditions. As these conditions change, the required off-body spacing changes as well. The viscous smoothing algorithm has the ability to quickly and efficiently change the off-body spacing or geometric progression of the viscous layers to accommodate changes to the flow characteristics with very little (or none, if the offset is coupled to the flow solution) manual adjustment to the mesh. Results are demonstrated below where the mesh starts out having inviscid properties and is then adjusted to accommodate different flow conditions.

The viscous mesh surrounding a NACA0012 airfoil that was seen in the previous examples of this section has an off-body spacing (Δs) of 0.010. For a y+ of 100 and a Reynolds number of 20,000 (and using a reference length of 1.0) this off-body spacing would be sufficient to yield 1 grid point in the laminar sublayer (Ref. 16) and thus make this an adequate mesh for examining a viscous flow in a very low-speed region using wall functions. The NACA0012 airfoil mesh at these conditions is shown below in Fig. 57.

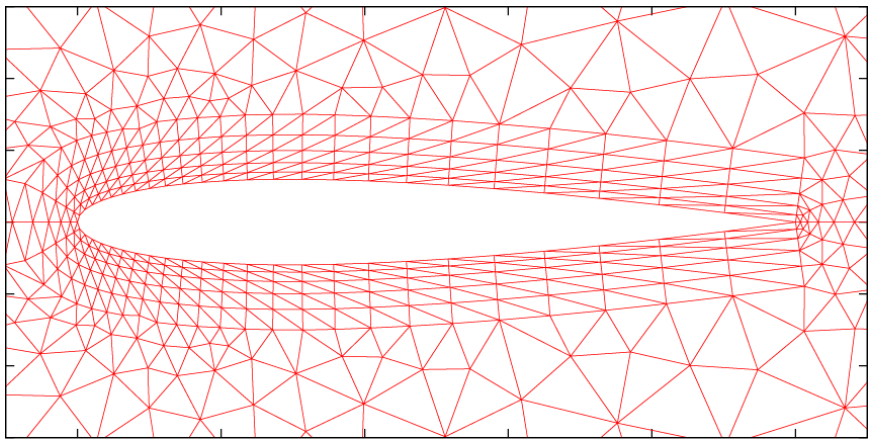

Figure 56. Mesh with Off-Body Spacing of $\Delta s = 0.01$ (y+ = 100, Re = 20,000)

The mesh shown in Fig. 57 would only be sufficient for capturing a boundary layer on extremely low-speed flows. If the speed was increased five-fold (resulting in a corresponding five times increase in the Reynolds number) the required off-body spacing for the same y+ (y+ = 100) would now be: $\Delta s = 0.00233$. Using the iteratively adapted computational space Winslow equations, no new mesh would need to be manually generated. A simple input change specifying the new off-body spacing would be sufficient to generate a new mesh with the required characteristics. (It might also be necessary to adjust relaxation factors to the smoother.) A mesh with characteristics necessary to analyze the flow at the new higher Reynolds number is shown in Fig. 58, and a close-up comparison of the two meshes, along with the original inviscid mesh, is shown in Fig. 59.

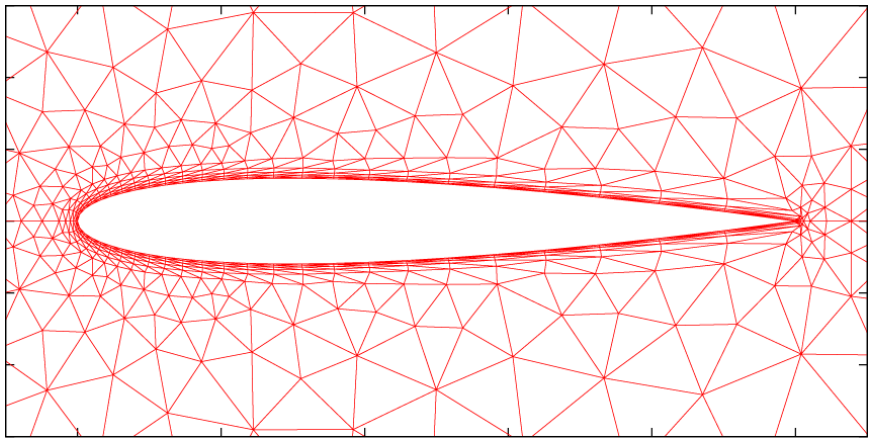

Figure 57. Mesh with Off-Body Spacing of Δs = 0.00233 (y+ = 100, Re = 100,000)

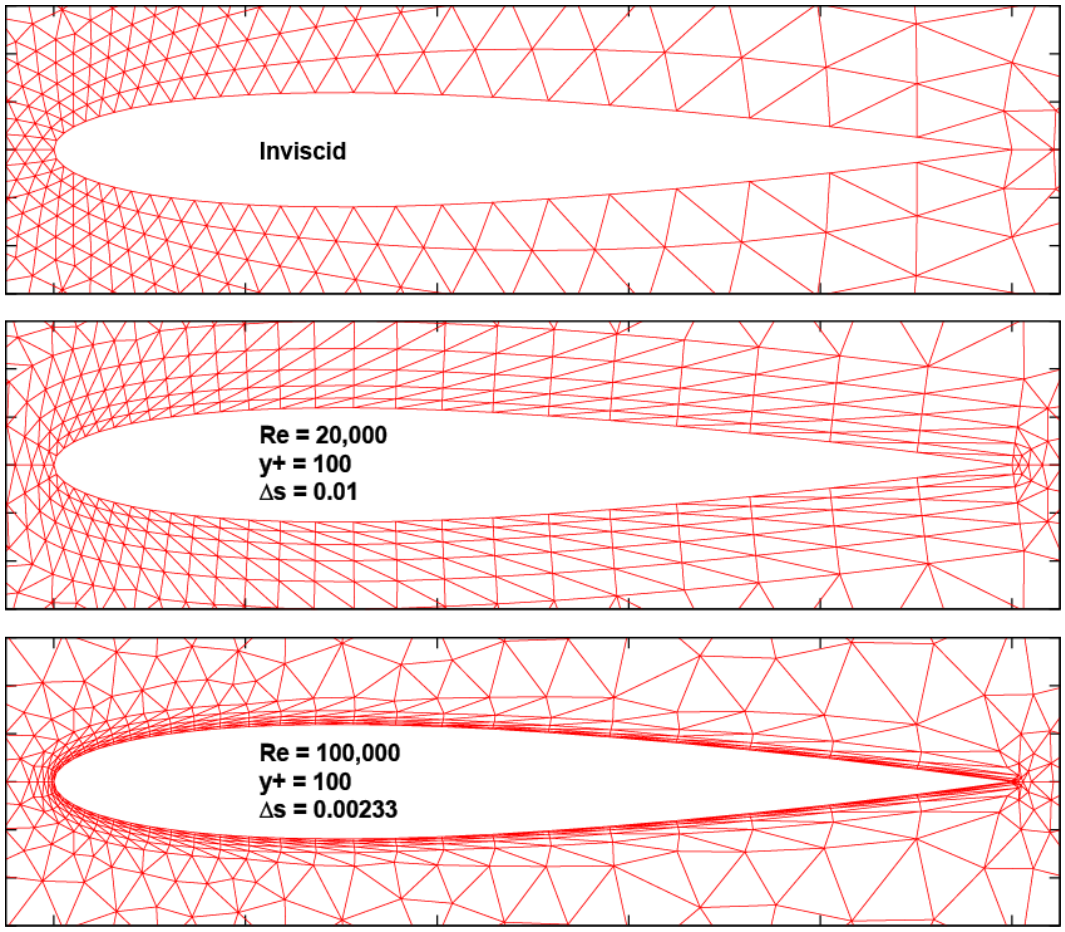

Figure 58. NACA0012 Mesh Comparison at Varying Reynolds Numbers

6.0 REFINEMENT

Adaptive mesh refinement (AMR) is utilized to manipulate a mesh in such a way that the mesh is able to effectively capture the required flowfield characteristics while keeping the number of mesh elements below the point where computational expense becomes prohibitive. AMR is carried out by adapting the mesh such that particular regions of the mesh are better suited to capture some important flow property or properties. This is done by refining the mesh in areas where the properties of interest have large gradients while keeping the size of the mesh elements relatively large in regions where the flow properties are not changing as rapidly with respect to space.

Three flowfield variables were examined for the purpose of refining (and also derefining) a mesh: pressure, velocity magnitude, and Mach number. The CFD code that was incorporated into the AMR process was a node-based solver. The flow variables are stored at the nodes, and the gradients are calculated using the technique described in Section 2.

6.1 UNSTRUCTURED ELEMENT DIVISION

The adaptation scheme that is implemented for the work described in this section is an edge-based scheme that is able to mark and refine the edges of an element in several ways. An adaptation function is used to determine which edges of which elements should be refined. Each edge consists of two nodes; a simple average of the gradients on the ends of the edge is not a favorable option because in the event where the gradients were high, but in opposing directions, they would cancel each other out. The adaptation function that is shown as Eq. (6.1) alleviates this problem.

$$Af = \frac{\left|\nabla \phi_1 \bullet \hat{r}\right| + \left|\nabla \phi_2 \bullet \hat{r}\right|}{2} l^p \tag{6.1}$$

In Eq. (6.1), $\nabla\phi_1$ is the gradient at the first node, $\nabla\phi_2$ is the gradient at the second node, $\hat{r}$ is the edge vector, and l^p is a parameter that is introduced to more efficiently handle length scales. If l^p is not included, regions in the flow where there are sharp features, such as a shock wave, will never reach a point where the adaptation function is not above some value where refinement is to take place. In fact, for a shock the gradient will actually increase as the cells are refined because the discontinuity remains but now occurs over a smaller distance. The ways in which an element can be marked are shown in Fig. 60, and the ways in which an element can be refined are shown in Fig. 61.

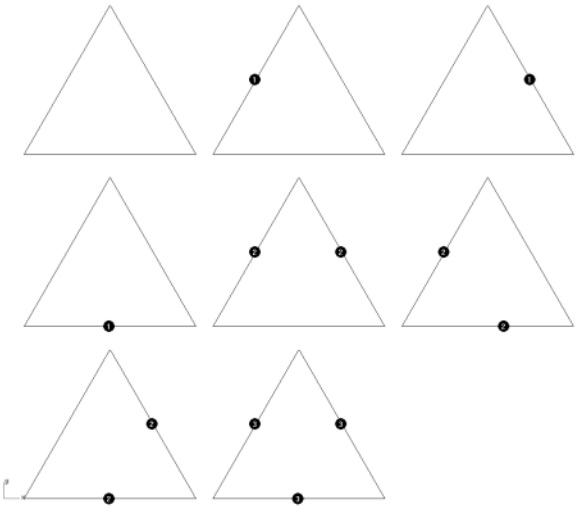

Figure 59. 8 (2^3) Ways to Mark a Triangle

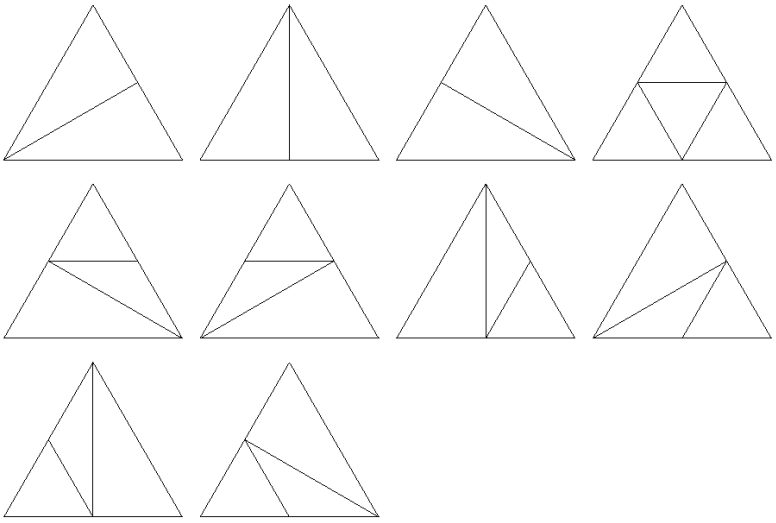

Figure 60. 11 Possible Ways to Split a Triangular Element

Because the refinement process is edge-based, there are 8 (i.e., 2^3) ways to mark a triangular mesh element and additional ways to split it up because not all of the markings result in a unique refinement. Note that in 3D there are 6 edges for a basic element (a tetrahedron), so there are 64 (i.e., 2^6) ways to mark the element and additional ways to split it up.

6.2 STATIC REFINEMENT

The first case that was examined for the purpose of demonstrating solution-based AMR was a transonic case involving a NACA0012 airfoil in a Mach 0.95 flow. At Mach 0.95 and zero angle of attack, the flow speed increases to about Mach 1.3 as it traverses the airfoil and then shocks down at the tail. However, the flow does not quite shock down to subsonic at that point and remains slightly supersonic for several chord lengths behind the airfoil. The flow then shocks down below the speed of sound, causing the airfoil to exhibit a distinct sonic line or “fishtail

shock" in its wake. This is illustrated in Fig. 62. The objective of examining this case was to adapt the original mesh (shown in Fig. 62) to the flow with the hope of capturing the shock waves coming off the airfoil and the secondary shock behind it.

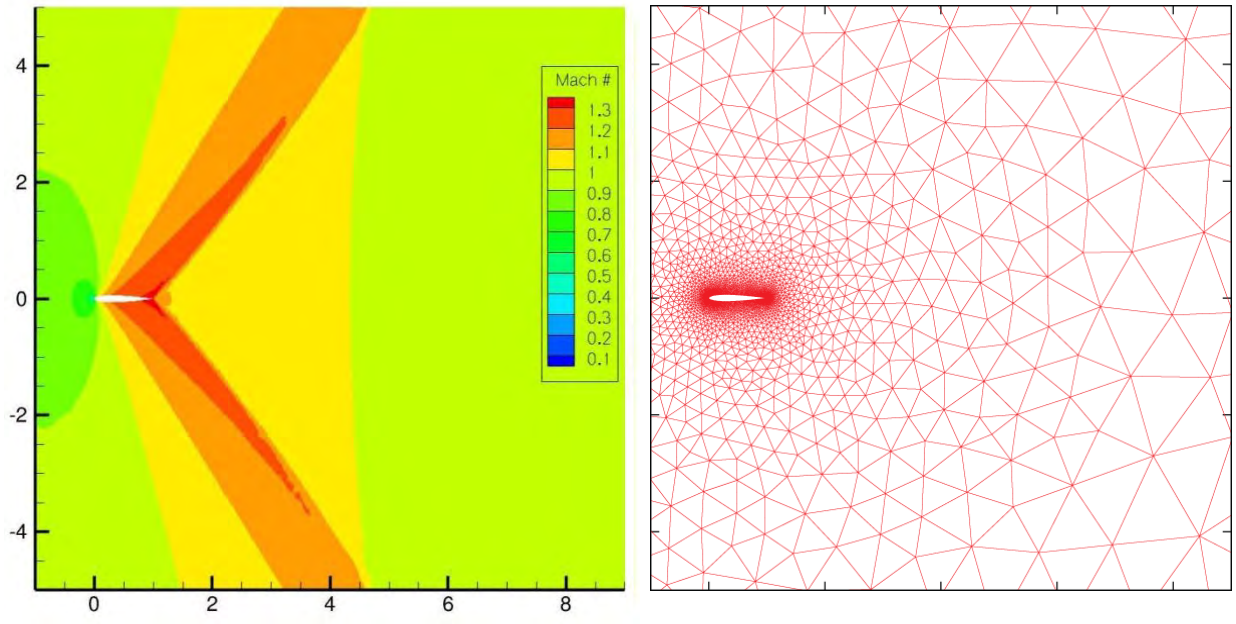

Figure 61. Flowfield Around a NACA0012 Airfoil at Mach 0.95 Along with the Initial Unrefined Mesh

Three different flow parameters were examined: Mach number, pressure, and velocity magnitude. Successful adaptation can be performed with all three parameters, but Mach number is focused on here because it captured some elements of both pressure and velocity. Shown below in the images comprising Fig. 63 is a progression of the grid over six refinement passes. Figure 62 was generated using Mach number as the flow parameter for which adaptation was performed. Figure 62 shows that the adaptation algorithm successfully captures the shock coming off the airfoil and the secondary shock behind it.

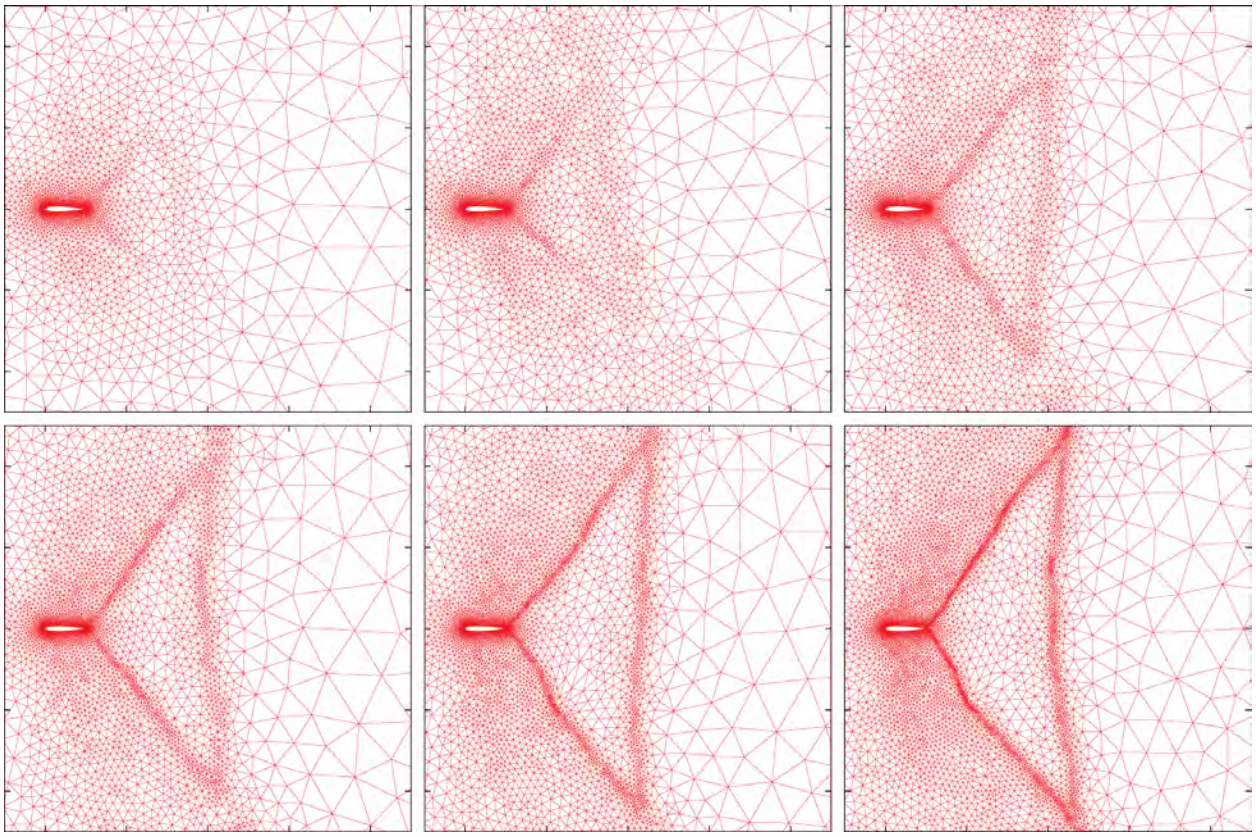

Figure 62. Progression of Grid Refinement/Derefinement

6.3 DYNAMIC REFINEMENT

After a steady-state case was performed, dynamic moving body cases were also examined. These cases used both AMR and elliptic smoothing to generate quality meshes at each step in the refinement process. In addition to refinement, derefinement was also implemented into the process with limited success. Derefinement is difficult to implement because, unlike refinement where overrefining a mesh may hurt performance but will probably not be significantly detrimental to a solution, aggressive derefinement can easily completely decimate a mesh to the point where no solution can be generated. However, derefinement can be important, as illustrated by the case described below which involves a NACA0012 airfoil at Mach 0.8.

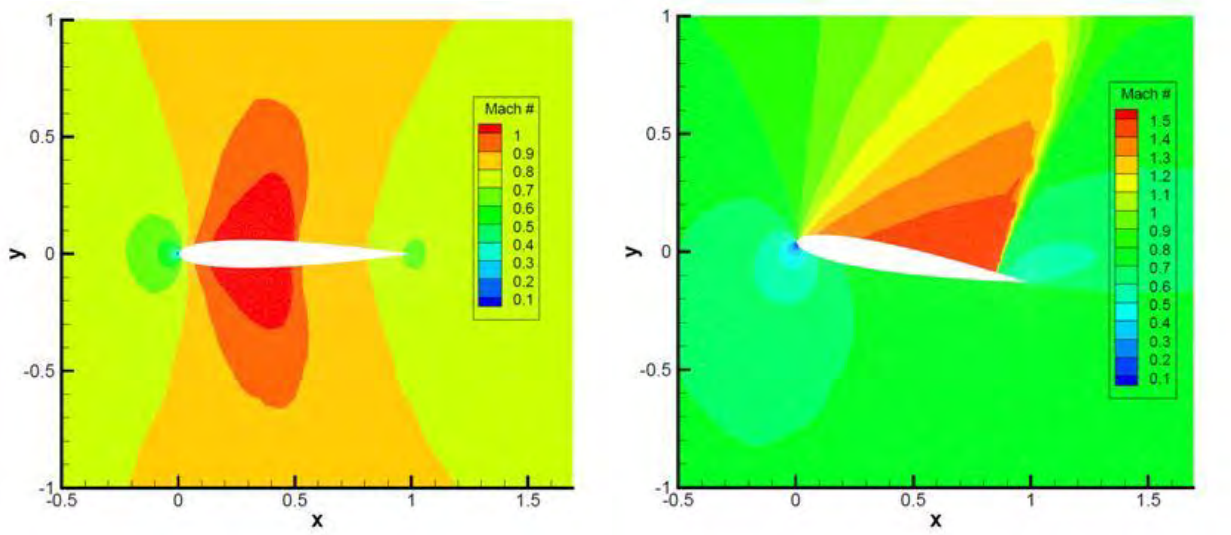

Figure 63. NACA0012 Airfoil at 0 deg Angle of Attack and at 10 deg Angle of Attack

The solutions to the flow over a NACA0012 airfoil at the two positions shown in Fig. 64 show that the shock moves significantly on the body of the airfoil when the airfoil is rotated. Figure 64 shows the case where the grid is adapted based on Mach number at zero angle of attack and the airfoil is then rotated -10 deg. At -10 deg angle of attack the shock is significantly further back on the body of the airfoil compared to the 0-deg case. If a solution on the initial mesh is used to perform adaptation and the airfoil is then rotated, Winslow smoothing will allow a valid mesh to be created for the new position, but if no derefinement is performed, the refinement from the 0-deg case will remain, even though the shock is no longer in this vicinity. The wasteful nature of this residual refinement is illustrated in Fig. 65.

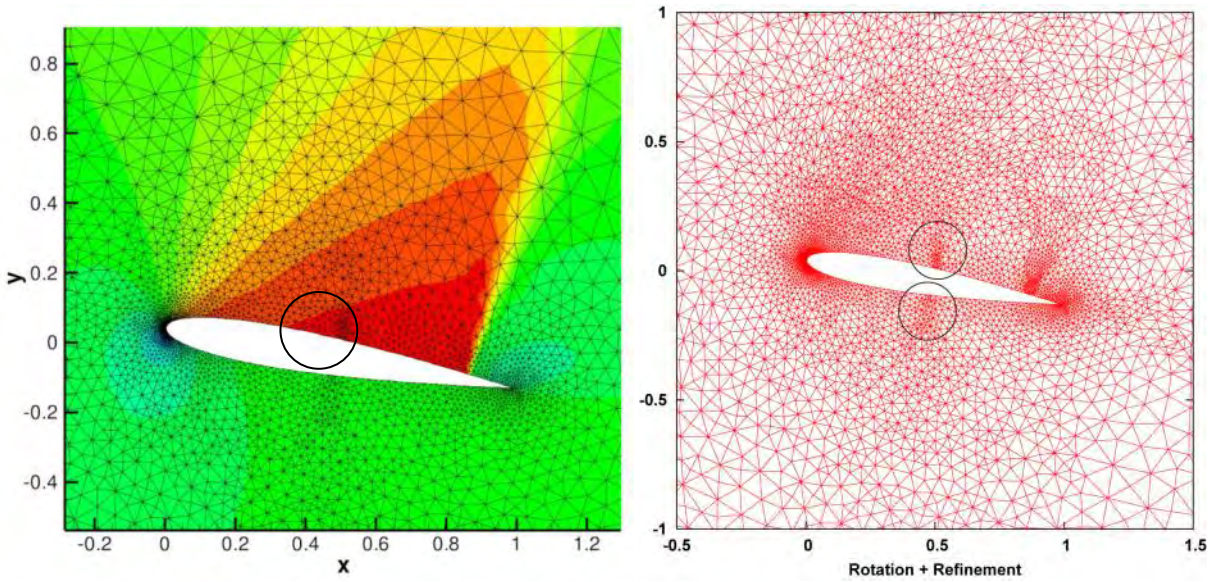

Figure 64. Grid Adaptation on Rotated Airfoil

The final case that was examined was that of a rotating airfoil. A NACA0012 airfoil, this time at a Mach number of 0.8, was rotated about the quarter chord, and the mesh was refined at each

rotation step. The airfoil was rotated up 1.5 deg, rotated back to zero, rotated down 1.5 deg, and then rotated back to the initial position.

The images in Fig. 66 show the refinement progression as the airfoil is pitched up and then down and then back up to the original position. For this case the derefinement capability was used very sparingly, and thus as expected there is a significant residual refinement as the angle of attack changes. More aggressive derefinement was tried, but the mesh in the upstream region, which had negligible gradients compared to the flow around the body, was always adversely affected. Despite the appearance of residual refinement and the associated degradation of computational efficiency, Fig. 66 shows that the shock is being well refined and that the smoothing technique is working well to rotate the interior points of the mesh as the airfoil surface boundary is rotated.

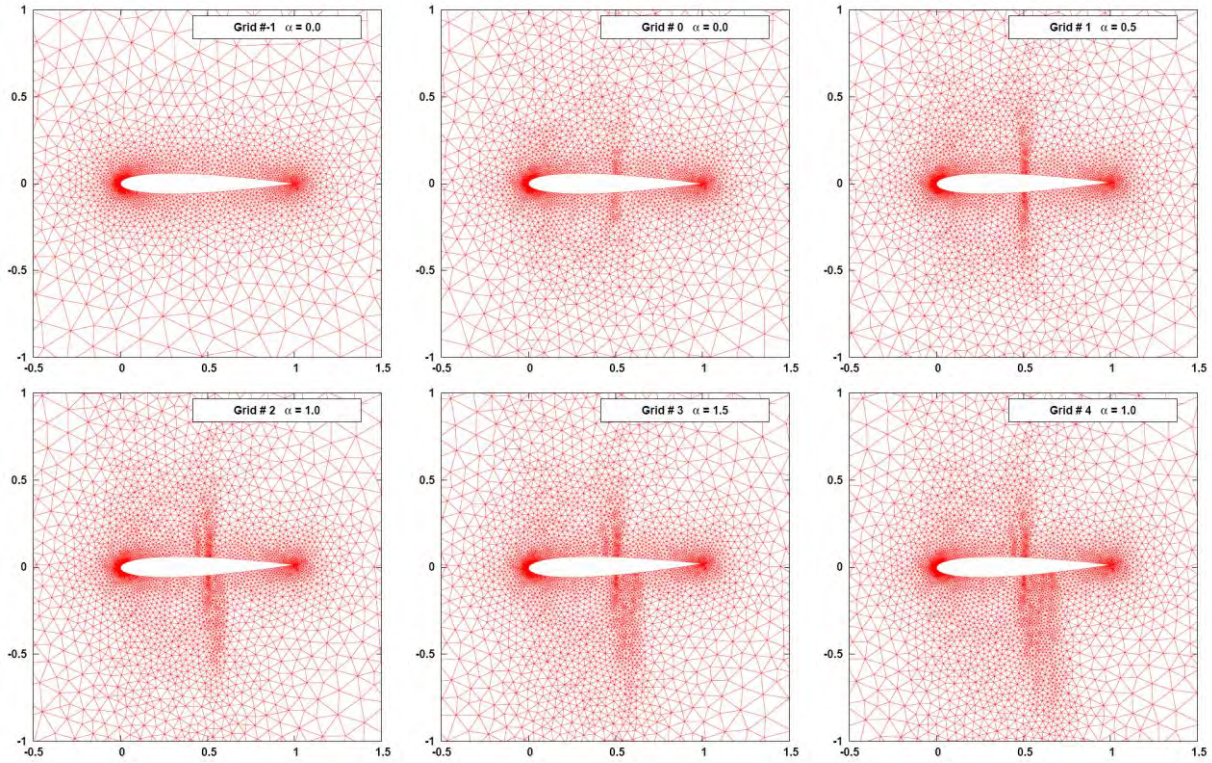

Figure 65. NACA0012 Refinement with Varying Pitch (a)

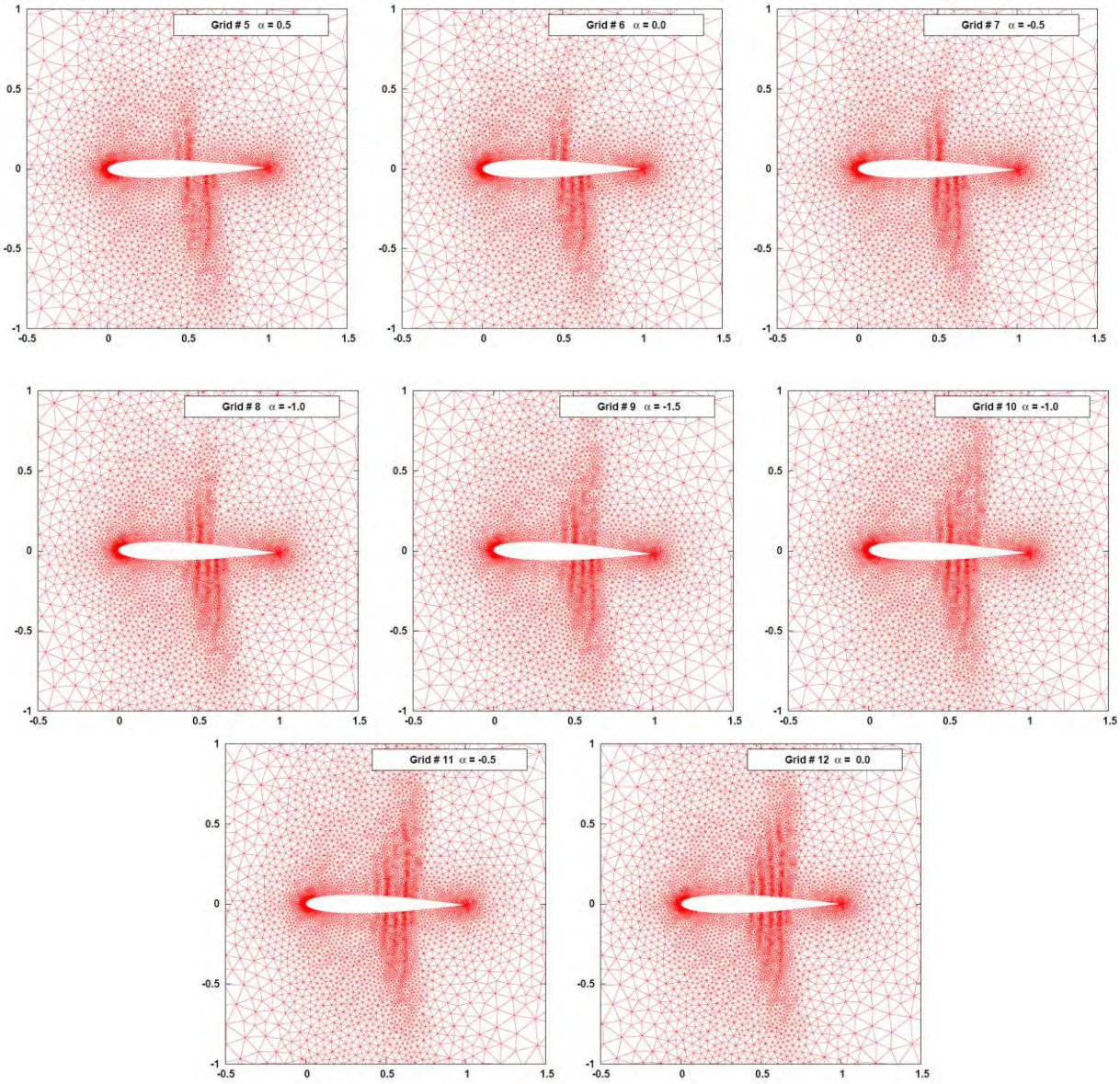

Figure 65. NACA0012 Refinement with Varying Pitch (b)

7.0 OPTIMIZATION

An efficient way to perform a design optimization on a 2D mesh is to couple an optimization code with a 2D CFD solver and a mesh movement/refinement code. The concept of optimization is based on finding a minimum of some function. There are several methods for finding minimums of a function; one such method is the Newton-Raphson technique. A design reaching an optimum state implies that the function for which the optimization has been performed has been minimized and the derivates of the function are zero. A simple one-variable example, such as that shown in Fig. 67, would be the function $f(x) = x^2 - 2$. The minimum can be found at $x = -2$ at which point $f'(x) = 0$. For a practical optimization, the function that is minimized is a cost function. The cost function, which is discussed below, will generally be a multivariate function driven by multiple design variables.

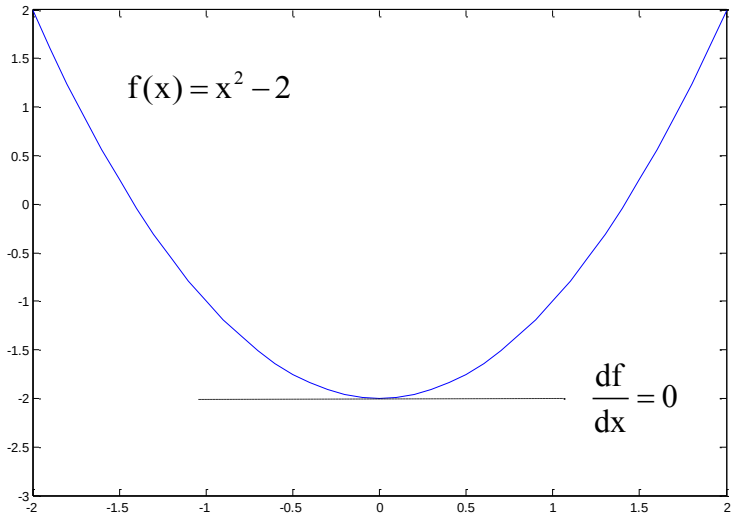

Figure 66. One Variable Example of a Function and Its Derivative

A common optimization method, such as used by the design code Port (developed by Bell Laboratories) and Dakota (developed by Sandia National Labs) is a trust region method. The trust region algorithm approximates only a certain region (the so-called trust region, which is locally approximated as quadratic) of a function as opposed to the entire function. When an adequate model of the function is found within the trust region, the region is expanded. The optimization tool used for the study described herein is Port. The source code for Port is freely available, and certain subroutines were stripped out to use directly with an optimization driver code that coupled the optimization tool with the CFD and mesh manipulation codes.

The optimization libraries from Port generally operate as black boxes (though the source is available to examine or modify if required). The specific port Library that was called by the driver routine was called DMNGB. This subroutine seeks a parameter vector (a vector of all of the design variables) that minimizes a continuously differentiable function (the cost function) subject to bound constraints. The caller provides a subroutine to compute the cost function and a second subroutine to compute the gradients of the cost function.

It is not the root of the function itself that is being sought, but rather roots of its gradients. To find these roots, a second derivative (Hessian matrix for a multivariate function) is necessary. Within the Port code, the Hessian (if it is not provided directly) is approximated by a secant (quasi-Newton) updating method.

7.1 COST FUNCTION

For the purposes of optimization, the function that is to be minimized is known as a cost function or an objective function. It is this function that is used to capture the relationship between the design variables and the design goals. The cases that were examined for this study were external flows over a naca0012 airfoil. The main parameters of interest were lift and drag. The cost function that was used to drive the design is shown as Eq. (7.1).

$$I = W_L \left(L - L^*\right)^2 + W_D \left(D - D^*\right)^2 \tag{7.1}$$

Where:

I	=	cost
W_L	=	lift coefficient (weight)
L	=	lift
L^*	=	target lift
W_D	=	drag coefficient (weight)
D	=	drag
D^*	=	target drag

7.2 DESIGN VARIABLES AND SENSITIVITY DERIVATIVES

A design variable is the aspect of the design that is being evaluated and modified to drive a design toward a certain behavior. There are infinite possibilities for what can be used as design variables. For an airfoil, typical design variables could be camber, thickness, angle of attack, or individual positions on the airfoil. It is common to denote the design variable as β_i.

For the study discussed here, the shape of the airfoil was manipulated directly by choosing the design variables to be the y coordinates of points on the airfoil surface. An important part of the optimization process is determining how each of the design variables affects the cost function. This is done through sensitivity derivatives, which are the partial derivatives of the cost with respect to each design variable (i.e., the change in the cost while perturbing a single design variable and keeping all of the other design variables constant).

The sensitivity derivatives are written as $\frac{\partial I}{\partial \beta_i}$.

Three methods were explored for generating sensitivity derivatives. They are:

- Finite Difference
- Forward Mode Differentiation
- Reverse Mode Differentiation

7.2.1 Sensitivity Derivatives Using Finite Difference

The most straightforward way to generate sensitivity derivatives is to use a finite difference approach. In this case, at each iteration (k) in the design process the cost function will be calculated using Eq. (7.2).

$$I_{(k)} = W_L \left(L_{(k)} - L^* \right)^2 + W_D \left(D_{(k)} - D^* \right)^2 \tag{7.2}$$

The derivatives of the cost function with respect to each of the design variables are found using the chain rule as:

$$\frac{\partial I}{\partial \beta_i} = 2 w_L \left(L - L^* \right) \frac{\partial L}{\partial \beta_i} + 2 w_D \left(D - D^* \right) \frac{\partial D}{\partial \beta_i}$$

Using the finite difference approach, each of the partial derivatives is found by perturbing β_i by a very small amount ($\Delta\beta$). The design derivatives are discretized and calculated as:

$$\frac{\partial I}{\partial \beta_{i\,(k)}} = 2.0 * w_L \left(L_{(k-1)} - L^*\right) \frac{L_{(k)} - L_{(k-1)}}{\Delta\beta_i} + 2.0 * w_D \left(D_{(k-1)} - D^*\right) \frac{D_{(k)} - D_{(k-1)}}{\Delta\beta_i}$$

This is the partial derivative with respect to a single design variable, so it must be performed for each design variable while holding all of the other design variables constant. This can be very computationally expensive because at each iteration in the design cycle, a flow solve is required to get the cost at iteration (k) and n mesh-movements, and n flow solves are required to generate the sensitivity derivatives where n is the number of design variables.

7.2.2 Sensitivity Derivatives Using Forward Mode Differentiation

As mentioned in the section above, a problem with using finite difference to get the sensitivity derivatives is that it requires so many CFD flow solves (n+1 where n is the number of design variables) per design cycle. One way to alleviate this requirement is to use forward mode differentiation (also known as direct differentiation or automatic differentiation). Using this method still requires the same amount of mesh movements at each design iteration, but requires only one full flow solve. For a case that is only concerned with lift, the cost function is:

$$I = C_L * \left(L - L^*\right)^2 + C_D * \left(D - D^*\right)^2 = 1.0 * \left(L - L^*\right)^2 + 0.0 * \left(D - D^*\right)^2$$
$$I = \left(L - L^*\right)^2$$

Lift, in turn, is the integral of vertical pressure forces over the entire wetted surface area of the airfoil and is calculated as the sum of the contributions to lift at each point on the airfoil, which is calculated using Eq. (7.3).

$$L = -|F|\, n_x \sin\alpha + |F|\, n_y \cos\alpha \tag{7.3}$$

In Eq. (7.3) α is the angle of attack of the airfoil, F is the force on the airfoil at a point, and n_x and n_y are the surface normals of the airfoil at that point. Cost is a function of lift, and lift is a function of other variables, which are either directly or indirectly related to the state variables (flow variables) or the physical geometry variables (mesh variables). The derivatives for the state variables and the mesh variables with respect to the design variables can be found directly by differentiating the code using the chain rule.

The starting point is the discrete residual of the system, which will be zero when the system is fully converged and the flow solution has reached a steady state. The residual of the system is a function of the design variables, the flow variables (which are functions of the design variables), and the mesh variables (which are also functions of the design variables). The residual of the system can be written as $R(Q(\beta), X(\beta), \beta) = 0$. Using the chain rule, the derivative of the residual for each of the design variables is:

$$\frac{dR}{d\beta_i} = \left\{\frac{\partial R}{\partial \beta_i}\right\} + \left[\frac{\partial R}{\partial Q}\right]\left\{\frac{\partial Q}{\partial \beta_i}\right\} + \left[\frac{\partial R}{\partial X}\right]\left\{\frac{\partial X}{\partial \beta_i}\right\} = 0 \tag{7.4}$$

Here, as well as throughout the remainder of this section, the derivatives in square brackets represent an array of values while the derivatives in curly brackets represent a vector of values. The partial derivative of the residual will be zero because all of the design variables are affecting the residual through the mesh and state vectors and not directly (through, for example, varying Mach number). This allows Eq. (7.4) to be rearranged into Eq. (7.5), which is the forward mode system equation.

$$\left[\frac{\partial \mathrm{R}}{\partial \mathrm{Q}}\right]\left\{\frac{\partial \mathrm{Q}}{\partial \beta_{\mathrm{i}}}\right\} = -\left[\frac{\partial \mathrm{R}}{\partial \mathrm{X}}\right]\left\{\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}\right\} \tag{7.5}$$

The derivates of the mesh node coordinates with respect to the design variables are referred to as the mesh sensitivities. These are dented as $\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}$ and are found by differentiating the mesh movement code. Once the mesh sensitivities are found, Eq. (7.5) is used to find $\frac{\partial \mathrm{Q}}{\partial \beta_{\mathrm{i}}}$. Once $\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}$ and $\frac{\partial \mathrm{Q}}{\partial \beta_{\mathrm{i}}}$ are known, the effect that the design variables have on any other variables in the system (effects on surface normals, lift, drag, etc.) can be found using the chain rule. Note that $\frac{\partial \mathrm{R}}{\partial \mathrm{Q}}$ is the partial derivative of the residual (i.e., the flux between elements in the mesh) with respect to the state vector while keeping the mesh constant. This is known because these values are the flux Jacobian values of the original mesh at steady state. $\frac{\partial \mathrm{R}}{\partial \mathrm{X}}$ is not found directly, but rather the entire right-hand side of Eq. (7.5) is found as $\left.\frac{\partial \mathrm{R}}{\partial \beta_{\mathrm{i}}}\right|_{\mathrm{Q\ fixed}}$ once $\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}$ is known.

7.2.3 Mesh Movement and the Complex Winslow Equations

In order to get the sensitivity derivatives, $\frac{\partial \mathrm{I}}{\partial \beta_{\mathrm{i}}}$, the mesh sensitivities, $\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}$, are needed. $\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}$ is a vector of equations for each design derivative. It will have a potentially different value at each point in the flowfield, and it represents the change in position of every mesh point in the flowfield caused by a change in one of the design variables. In 2D the expanded vector for the mesh sensitivities would look like Eq. (7.6).

$$\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}} = \begin{bmatrix} \frac{\partial \mathrm{x}}{\partial \beta_{\mathrm{i}}} \\ \frac{\partial \mathrm{y}}{\partial \beta_{\mathrm{i}}} \end{bmatrix} \tag{7.6}$$

One way that the mesh sensitivities can be found is by utilizing the mesh movement code. The design variables for this study were the y coordinates of points on the airfoil surface. In order to determine the mesh sensitivities, each of the design variables is tweaked by some small amount (ε) and the mesh movement code is run to see how each of the points in the flowfield responds.

One problem that can arise when generating mesh sensitivities in this manner is that when the perturbation is very small it can result in large subtractive cancelation error. For example, consider a case where a central difference was used. (For this example, this would be analogous to perturbing the design variable up by some small amount ($\beta = y + \Delta y$), down by some small amount ($\beta = y - \Delta y$), and then looking at how the points in the field reacted.)

$$\frac{\partial X}{\partial \beta_i} = \begin{bmatrix} \dfrac{x_{new} - x_{original}}{2\Delta y} \\ \dfrac{y_{new} - y_{original}}{2\Delta y} \end{bmatrix}$$

To illustrate the problem, consider the Taylor series expansion for a central difference scheme:

$$f(y + \Delta y) = f(y) + \frac{\partial f}{\partial y}\Delta y + \frac{\partial^2 f}{\partial y^2}\frac{(\Delta y)^2}{2!} + \frac{\partial^3 f}{\partial y^3}\frac{(\Delta y)^3}{3!} + \ldots$$

$$f(y - \Delta y) = f(y) - \frac{\partial f}{\partial y}\Delta y + \frac{\partial^2 f}{\partial y^2}\frac{(\Delta y)^2}{2!} - \frac{\partial^3 f}{\partial y^3}\frac{(\Delta y)^3}{3!} + \ldots$$

Subtracting the second equation from the first and ignoring higher order terms gives the following central difference scheme for a first derivative:

$$\frac{\partial f}{\partial y} = \frac{f(y + \Delta y) - f(y - \Delta y)}{2\Delta y} \tag{7.7}$$

Unfortunately, if Δy is very small, $y+\Delta y$ and $y-\Delta y$ look the same to the computer and error can quickly increase. Experience has shown that subtractive cancelation error starts to increase as Δy approaches the square root of machine zero (~E-8).

One way to reduce the subtractive cancelation error is to go into the complex plane. An alternate way of computing the first derivative of a function is to consider that function in complex space and manipulate the complex Taylor series expansion. If a function is perturbed by a small amount (ε) in the complex plane, the complex Taylor series is:

$$f(x + i\varepsilon) = f(x) + \frac{\partial f}{\partial x} i\varepsilon - \frac{\partial^2 f}{\partial x^2}\varepsilon^2 - \frac{\partial^3 f}{\partial x^3} i\frac{\varepsilon^3}{3!} + \ldots$$

Rearranging and ignoring higher order terms gives:

$$f(x + i\varepsilon) = \underbrace{f(x) - \frac{\partial^2 f}{\partial x^2}\varepsilon^2}_{\text{real part}} + \underbrace{\left[\frac{\partial f}{\partial x} h - \frac{\partial^3 f}{\partial x^3}\frac{\varepsilon^3}{3!}\right] i}_{\text{imaginary part}}$$

The derivative, then, can be calculated as:

$$\frac{\partial f}{\partial x}\varepsilon - \frac{\partial^3 f}{\partial x^3}\frac{\varepsilon^3}{3!} = \mathrm{Im}\left[f\left(x + i\varepsilon\right)\right]$$
$$\frac{\partial f}{\partial x} = \frac{\mathrm{Im}\left[f\left(x + i\varepsilon\right)\right]}{\varepsilon} + \vartheta\left(\varepsilon^2\right)$$

Ignoring the higher order terms gives the alternate formulation for calculating a first derivative of a function, which is shown as Eq. (7.8).

$$\frac{\partial f}{\partial x} = \frac{\mathrm{Im}\left[f\left(x + i\varepsilon\right)\right]}{\varepsilon} \tag{7.8}$$

In order to implement the alternated formulation, the mesh movement code is modified to operate in complex space and the Winslow smoother is extended to handle a complex (i.e., having a real and imaginary part) mesh. The mesh file is modified such that each node has a real and imaginary part for its x and y coordinate. The imaginary part of each of the nodes is zero with the exception of the node that would be used as the design variable. The design variable node now has a y coordinate that had an imaginary part ε.

When the mesh movement code is executed in complex mode, it no longer physically moves the mesh, but rather calculates a complex part for each of the coordinates. The complex part of each of the coordinates is divided by ε to generate the mesh sensitivities for a given design variable. If a point near the middle of the top surface of the airfoil is perturbed by a small amount ($d\beta = \varepsilon = 1.0\text{e-}10$) in the complex plane, the magnitude of the complex part of points in the flowfield can be seen (Fig. 68).

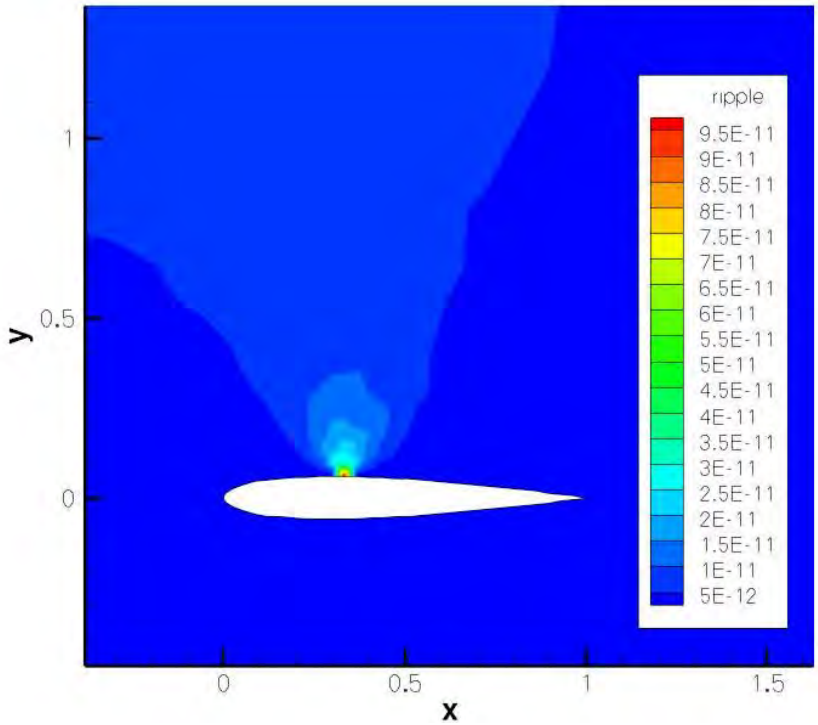

Figure 67. Magnitude of Mesh Sensitivities

Once the mesh sensitivities are known, the sensitivities of the state variables can be found by solving Eq. (7.5), and the sensitivity derivatives can be calculated and fed to Port to drive the optimization. For each design iteration, the mesh movement code must be executed a number of times equal to the number of the design variables, but a full CFD flow solve will only have to be executed once. For each design variable the mesh movement code will be executed to generate the mesh sensitivities, and then a linear system will be solved to generate the state vector sensitivities.

7.2.4 Sensitivity Derivatives Using Reverse Mode Differentiation

A third method for calculating the sensitivity derivatives is to use reverse mode differentiation. For forward mode differentiation, each of the design variables is perturbed, one at a time, and then the mesh movement code is executed to get mesh sensitivities. Once the mesh sensitivities are known for a given design variable, a linear system is solved to get the sensitivities of the state variables, and from this knowledge the sensitivity derivatives are found.

Forward mode differentiation yields the most information in that once the state variable sensitivities are determined, derivatives for anything else can be obtained. Although direct differentiation yields a significant amount of information, it can be very costly in cases where there are many design variables because Eq. (7.5) must be solved for each design variable. For cases where there are many design variables, a more efficient way can be to use reverse mode differentiation by employing adjoint methods.

For this method, the cost function is augmented so that the residual of the solver is included as a constraint using Lagrange multipliers. The cost function is a function of the state variables and the mesh: $\mathrm{I}(\mathrm{Q}(\beta),\mathrm{X}(\beta),\beta)$. The augmented equation is:

$$\mathrm{I}(\mathrm{Q}(\beta),\mathrm{X}(\beta),\beta) = \mathrm{I_c}(\mathrm{Q}(\beta),\mathrm{X}(\beta),\beta) + \Lambda^{\mathrm{T}}\mathrm{R}(\mathrm{Q}(\beta),\mathrm{X}(\beta),\beta)$$

In the equation above, $\mathrm{I_c}$ is the original cost and Λ is a vector of Lagrange multipliers (which are arbitrary). At steady state, because R = 0, $\mathrm{I} = \mathrm{I_c}$. Taking the derivative of I with respect to one of the design variables gives the augmented sensitivity derivative for a single design variable, which is shown as Eq. (7.9).

$$\frac{\mathrm{dI}}{\mathrm{d}\beta_{\mathrm{i}}} = \frac{\partial \mathrm{I_c}}{\partial \beta_{\mathrm{i}}} + \frac{\partial \mathrm{I_c}}{\partial \mathrm{Q}}\frac{\partial \mathrm{Q}}{\partial \beta_{\mathrm{i}}} + \frac{\partial \mathrm{I_c}}{\partial \mathrm{X}}\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}} + \Lambda^{\mathrm{T}}\left(\frac{\partial \mathrm{R}}{\partial \beta_{\mathrm{i}}} + \frac{\partial \mathrm{R}}{\partial \mathrm{Q}}\frac{\partial \mathrm{Q}}{\partial \beta_{\mathrm{i}}} + \frac{\partial \mathrm{R}}{\partial \mathrm{X}}\frac{\partial \mathrm{X}}{\partial \beta_{\mathrm{i}}}\right) \tag{7.9}$$

Equation (7.9) is the equation for a single design variable (β_{i}). The equation for the entire system can be manipulated to get:

$$\frac{\mathrm{dI}}{\mathrm{d}\beta} = \left[\frac{\partial \mathrm{Q}}{\partial \beta_{\mathrm{i}}}\right]^{\mathrm{T}}\left\{\frac{\partial \mathrm{I_c}}{\partial \mathrm{Q}} + \left[\frac{\partial \mathrm{R}}{\partial \mathrm{Q}}\right]^{\mathrm{T}}\Lambda\right\} + \left\{\frac{\partial \mathrm{I_c}}{\partial \beta} + \left[\frac{\partial \mathrm{X}}{\partial \beta}\right]^{\mathrm{T}}\frac{\partial \mathrm{I_c}}{\partial \mathrm{X}}\right\} + \left\{\left[\frac{\partial \mathrm{R}}{\partial \beta}\right]^{\mathrm{T}} + \left[\frac{\partial \mathrm{X}}{\partial \beta}\right]^{\mathrm{T}}\left[\frac{\partial \mathrm{R}}{\partial \mathrm{X}}\right]^{\mathrm{T}}\right\}\Lambda$$

$\frac{\partial Q}{\partial \beta_i}$ is unknown, but since Λ is arbitrary, $\left\{\frac{\partial I_c}{\partial Q}+\left[\frac{\partial R}{\partial Q}\right]^T \Lambda\right\}$ can be solved such that $\left[\frac{\partial Q}{\partial \beta_i}\right]^T \left\{\frac{\partial I_c}{\partial Q}+\left[\frac{\partial R}{\partial Q}\right]^T \Lambda\right\}$ is eliminated. This results in the adjoint equation, shown as Eq. (7.10).

$$\left[\frac{\partial R}{\partial Q}\right]^T \Lambda+\left\{\frac{\partial I_c}{\partial Q}\right\}=0 \tag{7.10}$$

Once the vector of Lagrange multipliers is determined, the sensitivity derivatives can be calculated using the augmented cost function for the entire system, shown as Eq. (7.11).

$$\frac{dI}{d\beta}=\left\{\frac{\partial I_c}{\partial \beta}+\left[\frac{\partial X}{\partial \beta}\right]^T \frac{\partial I_c}{\partial X}\right\}+\left\{\left[\frac{\partial R}{\partial \beta}\right]^T+\left[\frac{\partial X}{\partial \beta}\right]^T\left[\frac{\partial R}{\partial X}\right]^T\right\}\Lambda \tag{7.11}$$

For a case where all of the design variables are affecting the cost directly through the state and mesh variables, this equation can be simplified and written as Eq. (7.12).

$$\frac{dI}{d\beta}=\left[\frac{\partial X}{\partial \beta}\right]^T\left\{\frac{\partial I_c}{\partial X}\right\}+\Lambda^T\left[\frac{\partial R}{\partial \beta}\right]\Bigg|_{Q\ \text{fixed}} \tag{7.12}$$

The adjoint equation is a linear system; thus, for the cost of solving a linear system followed by a matrix-vector multiply for each design variable, the cost function with respect to all design variables can be obtained. Therefore, the adjoint method is efficient when there are many design variables and few cost function constraints (e.g., lift and drag).

7.3 SUMMARY OF SENSITIVITY DERIVATIVES

Sensitivity derivatives $\left(\frac{\partial I}{\partial \beta_i}\right)$ for the optimization can be found three different ways:

- Finite Difference
- Forward Mode Differentiation (AKA Direct Differentiation)
- Reverse Mode Differentiation (AKA Adjoint Method)

Finite difference is the most straightforward, but it is computationally the most expensive because for n design variables it requires n mesh movements and n+1 flow solves at each iteration in the design cycle.

Forward mode differentiation improves on the computational efficiency because at each iteration in the design cycle it requires n mesh movements but only requires one flow solve and n linear system solutions to solve $\left[\frac{\partial R}{\partial Q}\right]\left\{\frac{\partial Q}{\partial \beta_i}\right\}=-\left[\frac{\partial R}{\partial X}\right]\left\{\frac{\partial X}{\partial \beta_i}\right\}$ for $\frac{\partial Q}{\partial \beta_i}$.

If there are many design variables, this can still get expensive since the linear system is solved in an iterative fashion (using a point iterative scheme for this study) for each design variable, β_i.

The most computationally efficient way to solve for the sensitivity derivatives is to use reverse mode differentiation. In this case, adjoint methods are employed so that only one linear system solve is required. Reverse mode differentiation requires n mesh movements, one flow solve, and one linear system solve to solve $\left[\frac{\partial R}{\partial Q}\right]^T \Lambda = -\left\{\frac{\partial I_c}{\partial Q}\right\}$ for the vector of Lagrange multipliers (Λ). It also requires n matrix vector multiplies to solve:

$$\frac{dI}{d\beta} = \left\{\frac{\partial I_c}{\partial \beta} + \left[\frac{\partial X}{\partial \beta}\right]^T \frac{\partial I_c}{\partial X}\right\} + \left\{\left[\frac{\partial R}{\partial \beta}\right]^T + \left[\frac{\partial X}{\partial \beta}\right]^T \left[\frac{\partial R}{\partial X}\right]^T\right\} \Lambda$$

This will generally be more efficient than solving multiple linear systems unless the number of cost functions is much greater than the number of design variables.

8.0 CONCLUSIONS

There are many ways that a region that is to be analyzed using computational methods might be discretized. This report focused on breaking up a region of interest using unstructured grid methodology (as opposed to structured grid methodology, where the breakup of the domain reflects some type of consistent geometrical regularity). Although the focus was on unstructured meshes, many of the algorithmic and mathematical techniques found herein owe their origins to structured methods. The bulk of this report focused on unstructured mesh smoothing using the Winslow equations; adaptive mesh refinement and optimization were discussed as well.

The Winslow elliptic smoothing equations, which were first applied to structured meshes as far back as the 1960s, have started to become powerful tools for smoothing unstructured meshes as well. However, no implicit computational space exists for unstructured meshes, and the computational space must be explicitly defined. Because the unstructured mesh connectivity needs to be explicitly defined and the valence count (the number of connected nodes or neighbors) varies throughout a domain, no theory exists to generate a general overarching computational space that reflects a consistent geometrical regularity as was done with structured meshes. This difficulty was alleviated when it was determined that it was not necessary for the entire computational mesh to be constructed as an overarching system of nodes and elements, but rather each node in computational space could be treated as an individual virtual control volume and coupled only though the coordinates in physical space, which were treated as free dependent variables.

Using these loosely coupled virtual control volumes as the unstructured computational space, the Winslow elliptic smoothing equations allow interior mesh points to be moved and smoothed to conform to a moving surface or improve a static mesh. Conventionally, the unstructured computational space for each of the nodes formulated in this manner is surrounded by neighbors using equal angles and equal edge-lengths, and it is necessary that the central node in each virtual control volume be fully surrounded by neighboring nodes for the computational template to be complete. This makes the Winslow equations ideal for smoothing nonboundary nodes in inviscid meshes because the equations drive the mesh elements to display an isotropic behavior. However, the convention implementation of the Winslow equations is not well suited

for nodes on boundaries or nodes in viscous regions. Research detailed in this report addressed these limitations.

Traditionally, boundary points were held static in their original position, and only interior mesh points were permitted to move. However, if a moving surface was in close proximity to a static surface, the mesh quality could be significantly degraded. Because the connectivity of the mesh was not changing, the mesh elements were often not able to maintain low skew as they were stretched to conform to the new surface positions. The problem with applying the Winslow equations to a point on a surface is related to the virtual control volume of the surface node in computational space. For the equations to be properly solved, the virtual control volume needs to have a complete unbroken stencil of neighboring nodes. For a node on a surface, an unbroken stencil did not exist. This limitation was remedied by utilizing ghost points outside the proper physical mesh domain. Several methods for placing the ghost points were explored, and it was determined that the best results occurred by implementing a reflection technique, which reflected a composite interior-neighbor location across a line tangent to the boundary at the location of the boundary node for which a dynamic floating position was desired.

The floating-surface Winslow algorithm was tested on several geometries. The first case was a flat plate in a domain where the outer boundary was relatively close to the flat plate surface. The flat plate was taken through a full range of movements that included translation, rotation, and even the deformation of the surface. Valid meshes were generated for all cases. The final case that was examined to explore the effects of applying the Winslow equations to the boundaries was to apply Winslow elliptic smoothing to a 30P30N multi-element high-lift airfoil. The airfoil was pitched 30 deg, and the surrounding mesh was smoothed using the Winslow equations to adapt the flowfield mesh to the new airfoil surface position. The mesh was smoothed using both a static outer boundary and a floating-point outer boundary. Good results were generated and the smoothed mesh was tested further by using it to generate a transonic flow solution.

Because the goal of analyzing the 30P30N airfoil was to demonstrate the power of the Winslow equations applied to a boundary, it made sense to look at how two surfaces that were close together behaved if one of the surfaces was moved relative to the other. For this, the slat at the front of the airfoil was rotated from its original extended (high-lift) position to a retracted position. The mesh was smoothed at each of the two slat rotation steps. The first case that was examined was using static nodes on all of the airfoil boundaries. When this was done, the mesh became highly skewed in the region between the slat and the central airfoil position. The next case that was examined involved letting the boundary points on the surface of the central airfoil section float as the slat was rotated. This gave a much better mesh near the leading edge of the central airfoil section.

A conventional implementation of the Winslow equations involves constructing virtual control volumes for the computational space for each node using equal angles and equal edge-lengths. By implementing the computational space in this way, the Winslow equations smooth the physical mesh such that the modified (i.e., smoothed) mesh is highly isotropic and thus ideally suited for an inviscid flow. However, a viscous mesh near a no-slip wall is highly anisotropic and is thus in direct opposition to the goals of the unmodified Winslow equations. A successful methodology for applying the Winslow equations to viscous regions of unstructured meshes was to use an iteratively adapted computational space for each of the nodes in the viscous region. This technique allowed a mesh with an amenable connectivity structure to be smoothed in such a way that it could match a desired viscous profile based on an initial off-body spacing and geometric progression of the viscous layers.

The Winslow elliptic smoothing equations using an iteratively adaptive computational space algorithm were tested in situations where an airfoil (the NACA0012) was moved around within the domain. The airfoil was rotated, translated, and then both rotated and translated simultaneously. In any of the cases, the mesh can start out as viscous, inviscid, or even an unviable mesh with grid crossing and negative volumes. The Winslow equations cause the mesh to conform to the new position and orientation of the airfoil, and the offset computational space is able to maintain the viscous layer regardless of orientation. The technique was also explored on a deforming airfoil surface, which has many practical applications to parametric design where it may be required that many designs be examined, but manually generating meshes for each design would be prohibitively expensive. Again, a proper viscous mesh was generated.

A final application of the iteratively adaptive Winslow smoothing algorithm was a case where the required off-body spacing was changing. This is something that is guaranteed to be an issue in any kind of testing environment because it will always be necessary to explore properties at different Mach numbers and atmospheric conditions. As these conditions change, the required off-body spacing changes as well. It was shown that the iteratively adaptive Winslow smoothing algorithm has the ability to quickly and efficiently change the off-body spacing or geometric progression of the viscous layers to accommodate changes to the flow characteristics.

Adaptive mesh refinement (AMR) is utilized to manipulate a mesh in such a way that the mesh is able to effectively capture the required flowfield characteristics while keeping the number of mesh elements below the point where computational expense becomes prohibitive. AMR is carried out by modifying the mesh such that particular regions of the mesh are better suited to capture some important flow property or properties. This is done by refining the mesh in areas where the properties of interest have large gradients while keeping the size of the mesh elements relatively large in regions where the flow properties are not changing as rapidly with respect to space.

Three flowfield variables were examined for the purpose of refining (and also derefining) a mesh: pressure, velocity magnitude, and Mach number. AMR was explored on an airfoil that exhibited interesting “fishtail” shock characteristics, and the shocks were successfully captured. A dynamic case was also explored where the airfoil oscillated about the zero angle-of-attack line and the shock’s positions along the airfoil surface were changing. In the dynamic case, refinement was carried out successfully, but aggressive derefinement had a negative effect on grid quality, so the mesh became less efficient as it was continually refined.

A shape optimization study was performed in which an optimization code was coupled with a 2D CFD solver and a mesh movement/refinement code. The concept of optimization, which is based on finding a minimum of some function, was employed to drive the shape of an airfoil to exhibit desired characteristics. For this study, the shape of the airfoil was manipulated directly by choosing the design variables to be the y coordinates of points on the airfoil surface. Three methods were explored for generating sensitivity derivatives: finite difference, forward mode differentiation, and reverse mode differentiation. The differences in these three methods were discussed as well as the benefits and negative aspects of each. It was determined that under most circumstances the most computationally efficient method was to use reverse mode differentiation. In this case, adjoint methods are employed so that only one linear system solution is required.

The research and results described herein significantly expand the possible role of the Winslow elliptic smoothing equations as related to unstructured mesh manipulation. The methodologies

that were explored make the equations applicable to many common situations in fluid mechanics for which they traditionally would not have been well suited.

REFERENCES

1. Anderson, John D. *Computational Fluid Dynamics, The Basics with Applications.* New York, NY: MGraw-Hill, Inc., 1995. ISBN 0-07-001685-2.

2. Winslow, Alan M. “Numerical Solution of the Quasilinear Poisson Equations in a Nonuniform Triangle Mesh.” Worcester, MA: *Journal of Computational Physics*, 1967. Vol. 2, 149-172.

3. Sahasrabudhe, Mandar. “Unstructured Mesh Generation and Manipulation Based on Elliptic Smoothing and Optimization.” *A Dissertation Presented for the Doctorate of Philosophy Degree.* Chattanooga, TN: University of Tennessee at Chattanooga, 2008.

4. Karman, Steve and Sahasrabudhe, Mandar. “Unstructured Elliptic Smoothing Revisited.” *47th Aerospace Sciences Meeting and Exhibit.* Orlando, FL: American Institute of Aeronautics and Astronautics, 2009. AIAA-2009-1362.

5. Masters, James S. “Winslow Elliptic Smoothing Equations Extended to Apply to General Regions of an Unstructured Mesh.” *A Disertation Presented for the Doctor of Philosophy Degree.* Chattanooga, TN: University of Tennessee at Chattanooga, 2010.

6. Carpenter, James G. “Time Accurate Unstructured Grid Adaption in Two and Three Dimensions.” *A Dissertation Submitted to the Graduate Faculty of North Carolina State University.* Raleigh, NC: North Carolina State University, 2007.

7. Cavallo, Peter A and Boker, Timothy J. “Efficient Delaunay-Based Solution Adaptation for Three-Dimensional Unstructured Meshes.” Dublin, PA: American Institue of Aeronautics and Astronautics, 2000. AIAA-2000-0809.

8. Stewart, James. *Calculus, Third Edition.* Pacific Grove, CA: Brooks/Cole, 1995. ISBN 0-534-21798-2.

9. White, Frank M. *Viscous Fluid Flow.* Boston, MA: McGraw Hill, 1991. ISBN 0-07-069712-4.

10. Knupp, Patrick M. “Winslow Smoothing on Two-Dimensional Unstructured Meshes.” *Engineering With Computers.* Albuquerque, NM: Springer, 1999. Vol. 15.

11. Weisstein, Eric W. “Laplace's Equation.” From MathWorld - A Wolfram Web Resource. *http://mathworld.wolfram.com/LaplacesEquation.html.* 2010.

12. Karman, Steve, Anderson, Kyle, and Sahasrabudhe, Mandar. “Mesh Generation Using Unstructured Computational Meshes and Elliptic Partial Differential Equation Smoothing.” Reno, NV: American Institute of Aeronautics and Astronautics, 2005. AIAA-2005-0923.

13. www.pointwise.com. Fort Worth, TX: Pointwise, Inc., 2010.

14. Smith, Richard W. and Wright, Jeffrey A. “A Classical Elasticity-Based Mesh Update Method for Moving and Deforming Meshes.” Orlando, FL: American Institute of Aeronautics and Astronautics, 2010. AIAA-2010-164.

15. Babuska, I. and Aziz, A.K. “On the Angle Condition in the Finite Element Method.” *SIAM J. Numer. Anal.* Philadelphia, PA: Society for Industrial and Applied Mathematics, Apr. 1976. Vol. 13. pp 214-226.

16. Jones, William T. Gird Spacing Caculator. *http://geolab.larc.nasa.gov/APPS/YPlus/.* Langley, VA: NASA Langley Research Center, 2010.

17. Rowland, Todd. “Elliptic Partial Defferential Equation.” From MathWorld - A Wolfram Web Resource. *http://mathworld.wolfram.com/EllipticPartialDifferentialEquation.html.* 2010.

18. Thomas, P.D. and Middlecoff, J.F. “Direct Control of the Grid Point Distribution in Meshes Generated by Elliptic Equations.” *AIAA Journal.* Reston, VA: American Institute of Aeronautics and Astronautics, 1979. Vol. 18.

APPENDIX A. METRIC RELATIONSHIPS AND THE JACOBIAN

Partial derivatives with respect to the two-dimensional computational space coordinates (ξ,η) can be written as $\frac{\partial}{\partial\xi}=\left(\frac{\partial \mathrm{x}}{\partial\xi}\right)\left(\frac{\partial}{\partial \mathrm{x}}\right)+\left(\frac{\partial \mathrm{y}}{\partial\xi}\right)\left(\frac{\partial}{\partial \mathrm{y}}\right)$ and $\frac{\partial}{\partial\eta}=\left(\frac{\partial \mathrm{x}}{\partial\eta}\right)\left(\frac{\partial}{\partial \mathrm{x}}\right)+\left(\frac{\partial \mathrm{y}}{\partial\eta}\right)\left(\frac{\partial}{\partial \mathrm{y}}\right)$. The coefficients in these equations are referred to as the inverse grid metrics. Using a condensed form for the derivatives (i.e., $\frac{\partial \mathrm{x}}{\partial\xi}=\xi_{\mathrm{x}}$ etc.) the above relationships can be written as the following system:

$$\begin{bmatrix} \mathrm{x}_{\xi} & \mathrm{y}_{\xi} \\ \mathrm{x}_{\eta} & \mathrm{y}_{\eta} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \mathrm{x}} \\ \frac{\partial}{\partial \mathrm{y}} \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial\xi} \\ \frac{\partial}{\partial\eta} \end{bmatrix} \tag{A.1}$$

Cramer's rule allows one to construct the relationship between the grid metrics ($\xi_{\mathrm{x}}, \xi_{\mathrm{y}}, \eta_{\mathrm{x}}, \eta_{\mathrm{y}}$) and the inverse grid metrics ($\mathrm{x}_{\xi}, \mathrm{x}_{\eta}, \mathrm{y}_{\xi}, \mathrm{y}_{\eta}$). Using Cramer's rule, new relationships for the partial derivatives in physical space shown as Eqs. (A.2) and (A.3) are constructed.

$$\frac{\partial}{\partial \mathrm{x}}=\frac{\begin{vmatrix} \frac{\partial}{\partial\xi} & \mathrm{y}_{\xi} \\ \frac{\partial}{\partial\eta} & \mathrm{y}_{\eta} \end{vmatrix}}{\begin{vmatrix} \mathrm{x}_{\xi} & \mathrm{y}_{\xi} \\ \mathrm{x}_{\eta} & \mathrm{y}_{\eta} \end{vmatrix}} \tag{A.2}$$

$$\frac{\partial}{\partial \mathrm{y}}=\frac{\begin{vmatrix} \mathrm{x}_{\xi} & \frac{\partial}{\partial\xi} \\ \mathrm{x}_{\eta} & \frac{\partial}{\partial\eta} \end{vmatrix}}{\begin{vmatrix} \mathrm{x}_{\xi} & \mathrm{y}_{\xi} \\ \mathrm{x}_{\eta} & \mathrm{y}_{\eta} \end{vmatrix}} \tag{A.3}$$

In the equations above, the value in the denominator is the Jacobian, which is defined as:

$$\mathrm{J}=\begin{vmatrix} \mathrm{x}_{\xi} & \mathrm{y}_{\xi} \\ \mathrm{x}_{\eta} & \mathrm{y}_{\eta} \end{vmatrix}=\mathrm{x}_{\xi}\mathrm{y}_{\eta}-\mathrm{x}_{\eta}\mathrm{y}_{\xi} \tag{A.4}$$

Using the definition of the Jacobian, Eqs. (A.2) and (A.3) can now be written as follows:

$$\frac{\partial}{\partial \mathrm{x}}=\frac{1}{\mathrm{J}}\left(\mathrm{y}_{\eta}\frac{\partial}{\partial\xi}-\mathrm{y}_{\xi}\frac{\partial}{\partial\eta}\right) \tag{A.4}$$

$$\frac{\partial}{\partial \mathrm{y}} = \frac{1}{\mathrm{J}}\left(\mathrm{x}_{\xi} \frac{\partial}{\partial \eta} - \mathrm{x}_{\eta} \frac{\partial}{\partial \xi} \right) \tag{A.5}$$

Comparing Eqs. (A.4) and (A.5) to the chain-ruled equations for partial derivatives in physical space (i.e., $\frac{\partial}{\partial \mathrm{x}} = \xi_{\mathrm{x}} \frac{\partial}{\partial \xi} + \eta_{\mathrm{x}} \frac{\partial}{\partial \eta}$ and $\frac{\partial}{\partial \mathrm{y}} = \xi_{\mathrm{y}} \frac{\partial}{\partial \xi} + \eta_{\mathrm{y}} \frac{\partial}{\partial \eta}$) reveals the following relationships:

$$\xi_{\mathrm{x}} \frac{\partial}{\partial \xi} + \eta_{\mathrm{x}} \frac{\partial}{\partial \eta} = \frac{1}{\mathrm{J}}\left(\mathrm{y}_{\eta} \frac{\partial}{\partial \xi} - \mathrm{y}_{\xi} \frac{\partial}{\partial \eta} \right) \tag{A.6}$$

$$\xi_{\mathrm{y}} \frac{\partial}{\partial \xi} + \eta_{\mathrm{y}} \frac{\partial}{\partial \eta} = \frac{1}{\mathrm{J}}\left(\mathrm{x}_{\xi} \frac{\partial}{\partial \eta} - \mathrm{x}_{\eta} \frac{\partial}{\partial \xi} \right) \tag{A.7}$$

Combining Eqs. (A.6) and (A.7) into matrix form (shown as Eq. [A.8]) allows for the relationships between the grid metrics and the inverse grid metrics to be easily seen via inspection.

$$\begin{bmatrix} \xi_{\mathrm{x}} & \eta_{\mathrm{x}} \\ \xi_{\mathrm{y}} & \eta_{\mathrm{y}} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} = \frac{1}{\mathrm{J}} \begin{bmatrix} \mathrm{y}_{\eta} & -\mathrm{y}_{\xi} \\ -\mathrm{x}_{\eta} & \mathrm{x}_{\xi} \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial \xi} \\ \frac{\partial}{\partial \eta} \end{bmatrix} \tag{A.8}$$

$$\xi_{\mathrm{x}} = \frac{\mathrm{y}_{\eta}}{\mathrm{J}}$$

$$\eta_{\mathrm{x}} = -\frac{\mathrm{y}_{\xi}}{\mathrm{J}}$$

$$\xi_{\mathrm{y}} = -\frac{\mathrm{x}_{\eta}}{\mathrm{J}}$$

$$\eta_{\mathrm{y}} = \frac{\mathrm{x}_{\xi}}{\mathrm{J}}$$

$$\mathrm{J} = \mathrm{x}_{\xi}\mathrm{y}_{\eta} - \mathrm{x}_{\eta}\mathrm{y}_{\xi}$$

It is also necessary to define the inverse of this procedure. The partial derivatives of the physical variables, which are $\frac{\partial}{\partial \mathrm{x}} = \left(\frac{\partial \xi}{\partial \mathrm{x}} \right)\left(\frac{\partial}{\partial \xi} \right) + \left(\frac{\partial \eta}{\partial \mathrm{x}} \right)\left(\frac{\partial}{\partial \eta} \right)$ and $\frac{\partial}{\partial \mathrm{y}} = \left(\frac{\partial \xi}{\partial \mathrm{y}} \right)\left(\frac{\partial}{\partial \xi} \right) + \left(\frac{\partial \eta}{\partial \mathrm{y}} \right)\left(\frac{\partial}{\partial \eta} \right)$, can be written as the following system:

$$\begin{bmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{bmatrix} \begin{Bmatrix} \dfrac{\partial}{\partial \xi} \\ \dfrac{\partial}{\partial \eta} \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{Bmatrix} \tag{A.9}$$

Applying Cramer's rule to Eq. (A.9) gives the equations for the partial derivatives in computational space shown as Eqs. (A.10) and (A.11).

$$\frac{\partial}{\partial \xi} = \frac{\begin{vmatrix} \dfrac{\partial}{\partial x} & \eta_x \\ \dfrac{\partial}{\partial y} & \eta_y \end{vmatrix}}{\begin{vmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{vmatrix}} \tag{A.10}$$

$$\frac{\partial}{\partial \eta} = \frac{\begin{vmatrix} \xi_x & \dfrac{\partial}{\partial x} \\ \xi_y & \dfrac{\partial}{\partial y} \end{vmatrix}}{\begin{vmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{vmatrix}} \tag{A.11}$$

The value in the denominators of (A.10) and (A.11) is now the inverse Jacobian, which will be denoted with a script J ($\mathcal{J}$):

$$\mathrm{J} = \begin{vmatrix} \xi_x & \eta_x \\ \xi_y & \eta_y \end{vmatrix} = \xi_x \eta_y - \xi_y \eta_x \tag{A.12}$$

The equations for the partial derivatives in computational space above can now be written as follows:

$$\frac{\partial}{\partial \xi} = \frac{1}{\mathrm{J}} \left(\eta_y \frac{\partial}{\partial x} - \eta_x \frac{\partial}{\partial y} \right) \tag{A.13}$$

$$\frac{\partial}{\partial \eta} = \frac{1}{\mathrm{J}} \left(\xi_x \frac{\partial}{\partial y} - \xi_y \frac{\partial}{\partial x} \right) \tag{A.14}$$

Comparing Eqs. (A.13) and (A.14) to the equations for partial derivatives in computational space, i.e., $\dfrac{\partial}{\partial \xi} = \left(\dfrac{\partial x}{\partial \xi} \right) \left(\dfrac{\partial}{\partial x} \right) + \left(\dfrac{\partial y}{\partial \xi} \right) \left(\dfrac{\partial}{\partial y} \right)$ and $\dfrac{\partial}{\partial \eta} = \left(\dfrac{\partial x}{\partial \eta} \right) \left(\dfrac{\partial}{\partial x} \right) + \left(\dfrac{\partial y}{\partial \eta} \right) \left(\dfrac{\partial}{\partial y} \right)$, reveals the following relationships:

$$x_\xi \left(\frac{\partial}{\partial x} \right) + y_\xi \left(\frac{\partial}{\partial y} \right) = \frac{1}{J} \left(\eta_y \frac{\partial}{\partial x} - \eta_x \frac{\partial}{\partial y} \right) \tag{A.15}$$

$$x_\eta \left(\frac{\partial}{\partial x} \right) + y_\eta \left(\frac{\partial}{\partial y} \right) = \frac{1}{J} \left(\xi_x \frac{\partial}{\partial y} - \xi_y \frac{\partial}{\partial x} \right) \tag{A.16}$$

Combining Eqs. (A.15) and (A.16) into matrix form (shown as Eq. [A.17]) allows for the following relationships between the grid metrics and the inverse grid metrics (this time utilizing the inverse Jacobian) to again be easily seen via inspection.

$$\begin{bmatrix} x_\xi & y_\xi \\ x_\eta & y_\eta \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} = \frac{1}{J} \begin{bmatrix} \eta_y & -\eta_x \\ -\xi_y & \xi_x \end{bmatrix} \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \tag{A.17}$$

$$x_\xi = \frac{\eta_y}{J}$$

$$y_\xi = -\frac{\eta_x}{J}$$

$$x_\eta = -\frac{\xi_y}{J}$$

$$y_\eta = \frac{\xi_x}{J}$$

$$J = \xi_x \eta_y - \xi_y \eta_x$$

APPENDIX B. DERIVATION OF WINSLOW EQUATIONS

Elliptic smoothing can be used to generate and improve meshes in physical space. The equations that are used for the smoothing process are derived from the relationship that exists between a mesh in physical space and computational space. The Winslow equations were originally applied to structured meshes. A very simple structured mesh, in both physical and computational space, can be seen below.

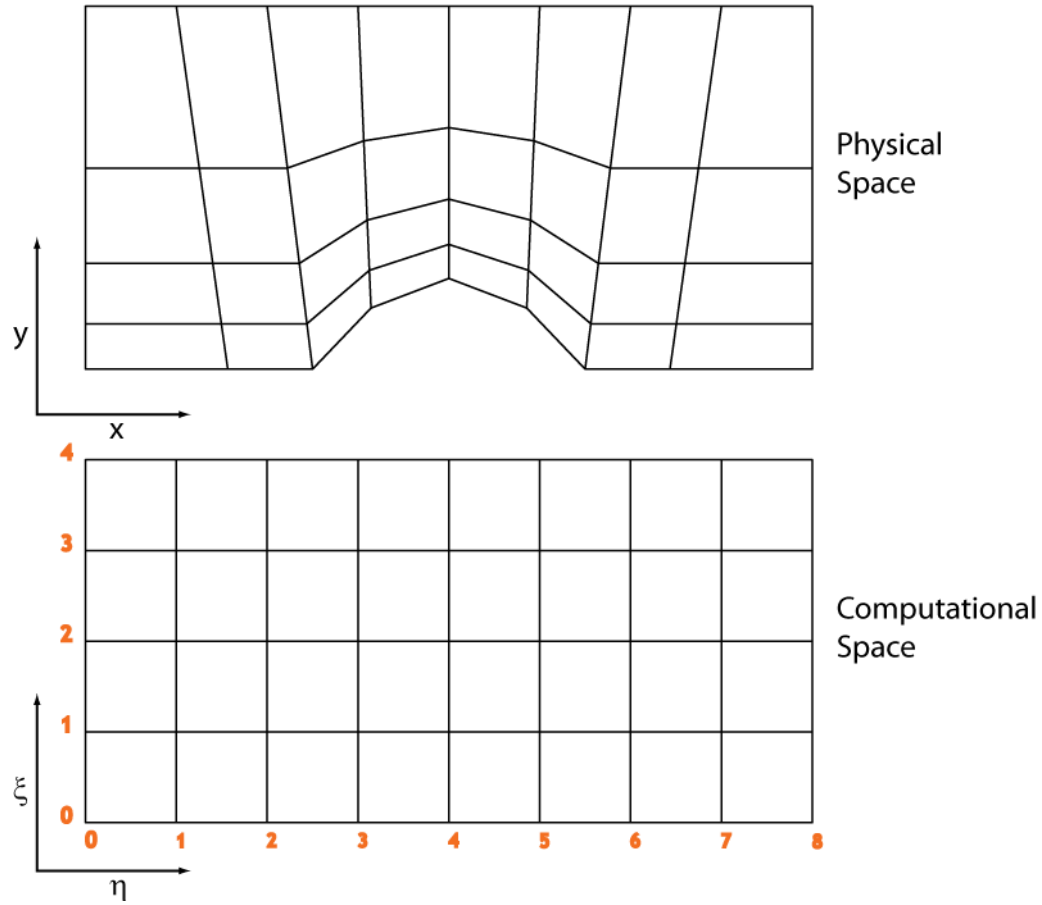

Figure B-1. Physical and Computational Space for a Structured Mesh

The coordinates in two-dimensional physical space are represented as (x,y), and the coordinates in two-dimensional computational space are represented as (ξ,η). The mapping between these two domains is defined with the transformations shown as Eqs. (B.1) and (B.2) (Ref. 12).

$$\begin{aligned} \xi &= \xi(x, y) \\ \eta &= \eta(x, y) \end{aligned} \tag{B.1}$$

$$\begin{aligned} x &= x(\xi, \eta) \\ y &= y(\xi, \eta) \end{aligned} \tag{B.2}$$

Elliptic smoothing utilizes elliptic partial differential equations in the form of Poisson's Eq. (B.3) or the homogeneous Laplace's Eq. (B.4) (Ref. 17).

$$\nabla^2\phi = \mathrm{f}(\mathrm{x},\mathrm{y}) \tag{B.3}$$

$$\nabla^2\phi = 0 \tag{B.4}$$

The elliptic smoothing equations are constructed by having a Laplacian operator with respect to physical coordinates act on the computational coordinates. The result is then set equal to zero if the Laplace form is desired and set equal to a forcing function if the Poisson form is desired. The elliptic smoothing equations are conventionally expressed in one of the two forms shown below:

P-Q Form:

$$\begin{aligned}\nabla^2\xi &= \frac{\partial^2\xi}{\partial \mathrm{x}^2} + \frac{\partial^2\xi}{\partial \mathrm{y}^2} = \mathrm{P} \\ \nabla^2\eta &= \frac{\partial^2\eta}{\partial \mathrm{x}^2} + \frac{\partial^2\eta}{\partial \mathrm{y}^2} = \mathrm{Q}\end{aligned} \tag{B.5}$$

$\Phi-\Psi$ Form:

$$\begin{aligned}\nabla^2\xi &= \frac{\partial^2\xi}{\partial \mathrm{x}^2} + \frac{\partial^2\xi}{\partial \mathrm{y}^2} = \left(\nabla\xi\cdot\nabla\xi\right)\Phi \\ \nabla^2\eta &= \frac{\partial^2\eta}{\partial \mathrm{x}^2} + \frac{\partial^2\eta}{\partial \mathrm{y}^2} = \left(\nabla\eta\cdot\nabla\eta\right)\Psi\end{aligned} \tag{B.6}$$

These equations are currently cast in physical space and represent the smooth variation of the computational coordinates in physical space. When the forcing functions are set equal to zero, the equations convert to Laplace's equations, which have the property that the average value over a spherical surface is equal to the value at the center of the sphere. This has the effect of causing a node to move to the centroid of the nodes to which it is connected. Solutions to Laplace's equations also satisfy the maximum/minimum principle, which states that the dependent variables on the interior of the domain are bounded by the values on the boundary of the domain. This ensures that the interior grid lines do not cross if the boundaries of the domain are chosen to be constant ξ and constant η gridlines (Ref. 12) as seen in Fig. B-1.

It was noted that the equations in the form shown in (B.5) and (B.6) represent a smooth variation of the computational coordinates in physical space. However, the computational coordinates (ξ and η) are generally known, and it is the values of the physical coordinates (x and y) that are desired. For the equations to be of practical value, they are transformed to computational space (i.e., the partial derivatives will be with respect to computational space). The final transformed equations are referred to as the Winslow equations, the derivation of which is shown below.

The chain rule for differentiation in multiple variables states that if $\mathrm{f} = \mathrm{f}(\xi, \eta)$ and $\xi = \xi(\mathrm{x}, \mathrm{y})$ and $\eta = \eta(\mathrm{x}, \mathrm{y})$, then the derivative for f is $\frac{\partial \mathrm{f}}{\partial \mathrm{x}} = \frac{\partial \mathrm{f}}{\partial \xi}\frac{\partial \xi}{\partial \mathrm{x}} + \frac{\partial \mathrm{f}}{\partial \eta}\frac{\partial \eta}{\partial \mathrm{x}}$. This can be rewritten in operator form as $\frac{\partial}{\partial \mathrm{x}}(\mathrm{f}) = \frac{\partial \xi}{\partial \mathrm{x}}\frac{\partial}{\partial \xi}(\mathrm{f}) + \frac{\partial \eta}{\partial \mathrm{x}}\frac{\partial}{\partial \eta}(\mathrm{f})$.

Now, consider the case where $\mathrm{f} = \frac{\partial \xi}{\partial \mathrm{x}}$. This gives the equation for the second derivative:

$$\frac{\partial}{\partial \mathrm{x}}\left(\frac{\partial \xi}{\partial \mathrm{x}}\right) = \frac{\partial \xi}{\partial \mathrm{x}}\frac{\partial}{\partial \xi}\left(\frac{\partial \xi}{\partial \mathrm{x}}\right) + \frac{\partial \eta}{\partial \mathrm{x}}\frac{\partial}{\partial \eta}\left(\frac{\partial \xi}{\partial \mathrm{x}}\right)$$

Or in condensed notation:

$$\xi_{\mathrm{xx}} = \xi_{\mathrm{x}}\xi_{\mathrm{x}\xi} + \eta_{\mathrm{x}}\xi_{\mathrm{x}\eta}$$

$$\xi_{\mathrm{xx}} = \xi_{\mathrm{x}}\frac{\partial}{\partial \xi}(\xi_{\mathrm{x}}) + \eta_{\mathrm{x}}\frac{\partial}{\partial \eta}(\xi_{\mathrm{x}})$$

$$\xi_{\mathrm{xx}} = \mathrm{J}^{-1}\mathrm{y}_{\eta}\frac{\partial}{\partial \xi}(\mathrm{J}^{-1}\mathrm{y}_{\eta})\xi_{\mathrm{x}} - \mathrm{J}^{-1}\mathrm{y}_{\xi}\frac{\partial}{\partial \eta}(\mathrm{J}^{-1}\mathrm{y}_{\eta})$$

$$\xi_{\mathrm{xx}} = \mathrm{J}^{-1}\mathrm{y}_{\eta}\left[-\mathrm{J}^{-2}\mathrm{J}_{\xi}\mathrm{y}_{\eta} + \mathrm{J}^{-1}\mathrm{y}_{\eta\xi}\right] - \mathrm{J}^{-1}\mathrm{y}_{\xi}\left[-\mathrm{J}^{-2}\mathrm{J}_{\eta}\mathrm{y}_{\eta} + \mathrm{J}^{-1}\mathrm{y}_{\eta\eta}\right]$$

$$\xi_{\mathrm{xx}} = \frac{\mathrm{y}_{\eta}}{\mathrm{J}^{3}}\left(\mathrm{J}\mathrm{y}_{\eta\xi} - \mathrm{J}_{\xi}\mathrm{y}_{\eta}\right) - \frac{\mathrm{y}_{\xi}}{\mathrm{J}^{3}}\left(\mathrm{J}\mathrm{y}_{\eta\eta} - \mathrm{J}_{\eta}\mathrm{y}_{\eta}\right) \tag{B.7}$$

Likewise:

$$\frac{\partial}{\partial \mathrm{x}}\left(\frac{\partial \eta}{\partial \mathrm{x}}\right) = \frac{\partial^{2}\eta}{\partial \mathrm{x}\partial \xi}\frac{\partial \xi}{\partial \mathrm{x}} + \frac{\partial^{2}\eta}{\partial \mathrm{x}\partial \eta}\frac{\partial \eta}{\partial \mathrm{x}} = \eta_{\xi\mathrm{x}}\xi_{\mathrm{x}} + \eta_{\eta\mathrm{x}}\eta_{\mathrm{x}}$$

$$\eta_{\mathrm{xx}} = \xi_{\mathrm{x}}\frac{\partial}{\partial \xi}(\eta_{\mathrm{x}}) + \eta_{\mathrm{x}}\frac{\partial}{\partial \eta}(\eta_{\mathrm{x}})$$

$$\eta_{\mathrm{xx}} = \mathrm{J}^{-1}\mathrm{y}_{\eta}\frac{\partial}{\partial \xi}(-\mathrm{J}^{-1}\mathrm{y}_{\xi})\xi_{\mathrm{x}} - \mathrm{J}^{-1}\mathrm{y}_{\xi}\frac{\partial}{\partial \eta}(-\mathrm{J}^{-1}\mathrm{y}_{\xi})$$

$$\eta_{\mathrm{xx}} = \mathrm{J}^{-1}\mathrm{y}_{\eta}\left[\mathrm{J}^{-2}\mathrm{J}_{\xi}\mathrm{y}_{\xi} - \mathrm{J}^{-1}\mathrm{y}_{\xi\xi}\right] - \mathrm{J}^{-1}\mathrm{y}_{\xi}\left[\mathrm{J}^{-2}\mathrm{J}_{\eta}\mathrm{y}_{\eta} - \mathrm{J}^{-1}\mathrm{y}_{\xi\eta}\right]$$

$$\eta_{\mathrm{xx}} = \frac{\mathrm{y}_{\eta}}{\mathrm{J}^{3}}\left(-\mathrm{J}\mathrm{y}_{\xi\xi} + \mathrm{J}_{\xi}\mathrm{y}_{\eta}\right) - \frac{\mathrm{y}_{\xi}}{\mathrm{J}^{3}}\left(-\mathrm{J}\mathrm{y}_{\xi\eta} + \mathrm{J}_{\eta}\mathrm{y}_{\eta}\right) \tag{B.8}$$

The derivatives of the Jacobian are as follows:

$$J_{\xi} = x_{\xi} y_{\eta\xi} + y_{\eta} x_{\xi\xi} - x_{\eta} y_{\xi\xi} - y_{\xi} x_{\eta\xi}$$
$$J_{n} = x_{\xi} y_{\eta\eta} + y_{\eta} x_{\eta\xi} - x_{\eta} y_{\xi\eta} - y_{\xi} x_{\eta\eta}$$

Substituting the values of the Jacobian and the Jacobian derivatives into (B.7) gives:

$$\xi_{xx} = \frac{y_{\eta}}{J^3}\left(x_{\xi} y_{\eta} y_{\eta\xi} - x_{\eta} y_{\xi} y_{\eta\xi} - x_{\xi} y_{\eta\xi} y_{\eta} - y_{\eta}{}^2 x_{\xi\xi} + x_{\eta} y_{\xi\xi} y_{\eta} + y_{\xi} x_{\eta\xi} y_{\eta}\right)$$
$$-\frac{y_{\xi}}{J^3}\left(x_{\xi} y_{\eta} y_{\eta\eta} - x_{\eta} y_{\xi} y_{\eta\eta} - x_{\xi} y_{\eta\eta} y_{\eta} - y_{\eta} x_{\eta\xi} y_{\eta} + x_{\eta} y_{\xi\eta} y_{\eta} + y_{\xi} x_{\eta\eta} y_{\eta}\right)$$

Rearranging and combining terms gives:

$$\xi_{xx} = \frac{1}{J^3}\left[x_{\eta}\left(y_{\eta}{}^2 y_{\xi\xi} + y_{\xi}{}^2 y_{\eta\eta} - 2 y_{\xi} y_{\eta} y_{\xi\eta}\right) - y_{\eta}\left(y_{\eta}{}^2 x_{\xi\xi} + y_{\xi}{}^2 x_{\eta\eta} - 2 y_{\xi} y_{\eta} x_{\eta\xi}\right)\right]$$

Following the same steps for the other terms gives:

$$\xi_{yy} = \frac{1}{J^3}\left[-y_{\eta}\left(x_{\eta}{}^2 x_{\xi\xi} + x_{\xi}{}^2 x_{\eta\eta} - 2 x_{\xi} x_{\eta} x_{\xi\eta}\right) + x_{\eta}\left(x_{\eta}{}^2 y_{\xi\xi} + x_{\xi}{}^2 y_{\eta\eta} - 2 x_{\xi} x_{\eta} y_{\eta\xi}\right)\right]$$

$$\eta_{xx} = -\frac{1}{J^3}\left[x_{\xi}\left(y_{\eta}{}^2 y_{\xi\xi} + y_{\xi}{}^2 y_{\eta\eta} - 2 y_{\xi} y_{\eta} y_{\xi\eta}\right) - y_{\xi}\left(y_{\eta}{}^2 x_{\xi\xi} + y_{\xi}{}^2 x_{\eta\eta} - 2 y_{\xi} y_{\eta} x_{\eta\xi}\right)\right]$$

$$\eta_{yy} = -\frac{1}{J^3}\left[-y_{\xi}\left(x_{\eta}{}^2 x_{\xi\xi} + x_{\xi}{}^2 x_{\eta\eta} - 2 x_{\xi} x_{\eta} x_{\xi\eta}\right) + x_{\xi}\left(x_{\eta}{}^2 y_{\xi\xi} + x_{\xi}{}^2 y_{\eta\eta} - 2 x_{\xi} x_{\eta} y_{\eta\xi}\right)\right]$$

Recall that the Laplacian of the computational coordinates are as follows:

$$\nabla^2 \xi = \xi_{xx} + \xi_{yy}$$
$$\nabla^2 \xi = \frac{1}{J^3}\left\{-y_{\eta}\left[\left(x_{\eta}{}^2 + y_{\eta}{}^2\right) x_{\xi\xi} - 2\left(x_{\xi} x_{\eta} + y_{\xi} y_{\eta}\right) x_{\xi\eta} + \left(x_{\xi}{}^2 + y_{\xi}{}^2\right) x_{\eta\eta}\right]\right.$$
$$\left. + x_{\eta}\left[\left(x_{\eta}{}^2 + y_{\eta}{}^2\right) y_{\xi\xi} - 2\left(x_{\xi} x_{\eta} + y_{\xi} y_{\eta}\right) y_{\xi\eta} + \left(x_{\xi}{}^2 + y_{\xi}{}^2\right) y_{\eta\eta}\right]\right\}$$

$$\nabla^2 \eta = \eta_{xx} + \eta_{yy}$$
$$\nabla^2 \eta = -\frac{1}{J^3}\left\{-y_{\xi}\left[\left(x_{\eta}{}^2 + y_{\eta}{}^2\right) x_{\xi\xi} - 2\left(x_{\xi} x_{\eta} + y_{\xi} y_{\eta}\right) x_{\xi\eta} + \left(x_{\xi}{}^2 + y_{\xi}{}^2\right) x_{\eta\eta}\right]\right.$$
$$\left. + x_{\xi}\left[\left(x_{\eta}{}^2 + y_{\eta}{}^2\right) y_{\xi\xi} - 2\left(x_{\xi} x_{\eta} + y_{\xi} y_{\eta}\right) y_{\xi\eta} + \left(x_{\xi}{}^2 + y_{\xi}{}^2\right) y_{\eta\eta}\right]\right\}$$

Breaking out the coefficients gives the Laplacians in computational coordinates in the following form:

$$\nabla^2\xi = \frac{1}{J^3}\left\{-y_\eta\left[\alpha x_{\xi\xi} - 2\beta x_{\eta\xi} + \gamma x_{\eta\eta}\right] + x_\eta\left[\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}\right]\right\}$$
$$\nabla^2\eta = -\frac{1}{J^3}\left\{-y_\xi\left[\alpha x_{\xi\xi} - 2\beta x_{\eta\xi} + \gamma x_{\eta\eta}\right] + x_\xi\left[\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta}\right]\right\} \quad \text{(B.9)}$$

where
$$\alpha = x_\eta^{\,2} + y_\eta^{\,2}$$
$$\beta = x_\xi x_\eta + y_\xi y_\eta$$
$$\gamma = x_\xi^{\,2} + y_\xi^{\,2}$$

Now an operator is defined $G(\square)$ as:

$$G(\square) = \alpha\frac{\partial^2(\square)}{\partial\xi^2} - 2\beta\frac{\partial^2(\square)}{\partial\xi\partial\eta} + \gamma\frac{\partial^2(\square)}{\partial\eta^2} \quad \text{(B.10)}$$

Using the operator G, Eqs. (B.9) now become:

$$\nabla^2\xi = \frac{1}{J^3}\left(-y_\eta G(x) + x_\eta G(y)\right) = -\frac{1}{J^2}\left(\xi_x G(x) + \xi_y G(y)\right)$$

$$\nabla^2\eta = -\frac{1}{J^3}\left(-y_\xi G(x) + x_\xi G(y)\right) = \frac{1}{J^2}\left(\eta_x G(x) + \eta_y G(y)\right)$$

Define a vector $\vec{r} = \begin{bmatrix} x \\ y \end{bmatrix}$ and these become:

$\nabla^2\xi = -\frac{1}{J^2}\left(\nabla\xi \cdot G(\vec{r})\right)$ and $\nabla^2\eta = -\frac{1}{J^2}\left(\nabla\eta \cdot G(\vec{r})\right)$

Rearranging the above equations slightly gives:

$$\left(\xi_x G(x) + \xi_y G(y)\right) = -J^2\nabla^2\xi$$
$$\left(\eta_x G(x) + \eta_y G(y)\right) = -J^2\nabla^2\eta$$

In Matrix form this is:

$$\begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix}\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = -J^2\begin{bmatrix} \nabla^2\xi \\ \nabla^2\eta \end{bmatrix}$$

Solving for G(x) and G(y) gives:

$$\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = -J^2 \begin{bmatrix} \xi_x & \xi_y \\ \eta_x & \eta_y \end{bmatrix}^{-1} \begin{bmatrix} \nabla^2 \xi \\ \nabla^2 \eta \end{bmatrix}$$

$$\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = \frac{-J^2}{\xi_x \eta_y - \xi_y \eta_x} \begin{bmatrix} \eta_y & -\xi_y \\ -\eta_x & \xi_x \end{bmatrix} \begin{bmatrix} \nabla^2 \xi \\ \nabla^2 \eta \end{bmatrix}$$

Using the inverse-Jacobian, derived in Appendix A and shown as (A.12), this becomes:

$$\begin{bmatrix} G(x) \\ G(y) \end{bmatrix} = \frac{-J^2}{J} \begin{bmatrix} \eta_y \nabla^2 \xi - \xi_y \nabla^2 \eta \\ -\eta_x \nabla^2 \xi + \xi_x \nabla^2 \eta \end{bmatrix}$$

$$G(x) = -J^2 \left[\frac{\eta_y}{J} \nabla^2 \xi - \frac{\xi_y}{J} \nabla^2 \eta \right] = -J^2 \left[x_\xi \nabla^2 \xi + x_\eta \nabla^2 \eta \right] \tag{B.11}$$

$$G(y) = -J^2 \left[-\frac{\eta_x}{J} \nabla^2 \xi + \frac{\xi_x}{J} \nabla^2 \eta \right] = -J^2 \left[y_\xi \nabla^2 \xi + y_\eta \nabla^2 \eta \right] \tag{B.12}$$

B.1 ZERO FORCING FUNCTION

The desired conditions for smooth interior points using the homogeneous Laplace's equation is $\nabla^2 \xi = 0$ and $\nabla^2 \eta = 0$. If $\nabla^2 \eta$ and $\nabla^2 \xi$ are set to zero in Eqs. (B.11) and (B.12), then it follows that G(x) = 0 and G(y) = 0. Using the definition for G(x) and G(y) shown in Eq. (B.10) gives the final form of the Winslow equations with no forcing function:

$$G(x) = 0$$

$$G(y) = 0$$

$$\begin{aligned} \alpha \frac{\partial^2 x}{\partial \xi^2} - 2\beta \frac{\partial^2 x}{\partial \xi \eta} + \gamma \frac{\partial^2 x}{\partial \eta^2} = 0 \\ \alpha \frac{\partial^2 y}{\partial \xi^2} - 2\beta \frac{\partial^2 y}{\partial \xi \eta} + \gamma \frac{\partial^2 y}{\partial \eta^2} = 0 \end{aligned} \tag{B.13}$$

where:

$$\alpha = x_\eta{}^2 + y_\eta{}^2$$

$$\beta = x_\xi x_\eta + y_\xi y_\eta$$

$$\gamma = x_\xi{}^2 + y_\xi{}^2$$

B.2 WINSLOW EQUATIONS WITH FORCING FUNCTIONS

Consider Eqs. (B.5) and (B.6) with the forcing functions (P and Q) intact (i.e., the nonhomogeneous Poisson's equation form):

$$\nabla^2\xi = \mathrm{P}(\mathrm{x},\mathrm{y})$$
$$\nabla^2\eta = \mathrm{Q}(\mathrm{x},\mathrm{y})$$

The Laplacians of the computational coordinates can now again be related to $\mathrm{G(x)}$ and $\mathrm{G(y)}$ by examining Eqs. (B.11) and (B.12). (Note that at this point no assumptions were made as to whether the equations were in the Laplace or Poisson form.)

$$\mathrm{G(x)} = -\mathrm{J}^2\left[\mathrm{x}_\xi\nabla^2\xi + \mathrm{x}_\eta\nabla^2\eta\right]$$

$$\mathrm{G(y)} = -\mathrm{J}^2\left[\mathrm{y}_\xi\nabla^2\xi + \mathrm{y}_\eta\nabla^2\eta\right]$$

Applying the forcing functions gives:

$$\mathrm{G(x)} = -\mathrm{J}^2\left[\mathrm{x}_\xi\mathrm{P} + \mathrm{x}_\eta\mathrm{Q}\right]$$
$$\mathrm{G(y)} = -\mathrm{J}^2\left[\mathrm{y}_\xi\mathrm{P} + \mathrm{y}_\eta\mathrm{Q}\right]$$

Thomas and Middlecoff (Ref. 18) proposed the following form for the forcing functions:

$$\begin{aligned}\mathrm{P}(\mathrm{x},\mathrm{y}) &= \Phi(\xi,\eta)(\nabla\xi\cdot\nabla\xi)\\ \mathrm{Q}(\mathrm{x},\mathrm{y}) &= \Psi(\xi,\eta)(\nabla\eta\cdot\nabla\eta)\end{aligned} \qquad \text{(B.14)}$$

Using the Thomas-Middlecoff form (a.k.a. the Φ-Ψ form) gives:

$$\mathrm{G(x)} = -\mathrm{J}^2\left[\mathrm{x}_\xi\Phi(\nabla\xi\cdot\nabla\xi) + \mathrm{x}_\eta\Psi(\nabla\eta\cdot\nabla\eta)\right]$$
$$\mathrm{G(y)} = -\mathrm{J}^2\left[\mathrm{y}_\xi\Phi(\nabla\xi\cdot\nabla\xi) + \mathrm{y}_\eta\Psi(\nabla\eta\cdot\nabla\eta)\right]$$

Employing the definition of the G operator from Eq. (B.10) results in:

$$\alpha\mathrm{x}_{\xi\xi} - 2\beta\mathrm{x}_{\xi\eta} + \gamma\mathrm{x}_{\eta\eta} = -\mathrm{J}^2\left[\mathrm{x}_\xi\Phi(\nabla\xi\cdot\nabla\xi) + \mathrm{x}_\eta\Psi(\nabla\eta\cdot\nabla\eta)\right]$$
$$\alpha\mathrm{y}_{\xi\xi} - 2\beta\mathrm{y}_{\xi\eta} + \gamma\mathrm{y}_{\eta\eta} = -\mathrm{J}^2\left[\mathrm{y}_\xi\Phi(\nabla\xi\cdot\nabla\xi) + \mathrm{y}_\eta\Psi(\nabla\eta\cdot\nabla\eta)\right]$$

$$\begin{aligned}\alpha\mathrm{x}_{\xi\xi} - 2\beta\mathrm{x}_{\xi\eta} + \gamma\mathrm{x}_{\eta\eta} + \mathrm{J}^2\mathrm{x}_\xi\Phi(\nabla\xi\cdot\nabla\xi) + \mathrm{J}^2\mathrm{x}_\eta\Psi(\nabla\eta\cdot\nabla\eta) &= 0\\ \alpha\mathrm{y}_{\xi\xi} - 2\beta\mathrm{y}_{\xi\eta} + \gamma\mathrm{y}_{\eta\eta} + \mathrm{J}^2\mathrm{y}_\xi\Phi(\nabla\xi\cdot\nabla\xi) + \mathrm{J}^2\mathrm{y}_\eta\Psi(\nabla\eta\cdot\nabla\eta) &= 0\end{aligned} \qquad \text{(B.15)}$$

Applying the dot product to the computational coordinates gives:

$$\nabla\xi\cdot\nabla\xi = \xi_\mathrm{x}\xi_\mathrm{x} + \xi_\mathrm{y}\xi_\mathrm{y} = \frac{\mathrm{y}_\eta}{\mathrm{J}}\frac{\mathrm{y}_\eta}{\mathrm{J}} + \frac{-\mathrm{x}_\eta}{\mathrm{J}}\frac{-\mathrm{x}_\eta}{\mathrm{J}} = \frac{1}{\mathrm{J}^2}\left(\mathrm{y}_\eta{}^2 + \mathrm{x}_\eta{}^2\right)$$
$$\nabla\eta\cdot\nabla\eta = \eta_\mathrm{x}\eta_\mathrm{x} + \eta_\mathrm{y}\eta_\mathrm{y} = \frac{-\mathrm{y}_\xi}{\mathrm{J}}\frac{-\mathrm{y}_\xi}{\mathrm{J}} + \frac{\mathrm{x}_\xi}{\mathrm{J}}\frac{\mathrm{x}_\xi}{\mathrm{J}} = \frac{1}{\mathrm{J}^2}\left(\mathrm{y}_\xi{}^2 + \mathrm{x}_\xi{}^2\right)$$

This is now plugged into Eq. (B.15).

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} + x_{\xi}\Phi\left(y_{\eta}^{\,2} + x_{\eta}^{\,2}\right) + x_{\eta}\Psi\left(y_{\xi}^{\,2} + x_{\xi}^{\,2}\right) = 0$$
$$\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} + y_{\xi}\Phi\left(y_{\eta}^{\,2} + x_{\eta}^{\,2}\right) + y_{\eta}\Psi\left(y_{\xi}^{\,2} + x_{\xi}^{\,2}\right) = 0$$

$$\alpha x_{\xi\xi} - 2\beta x_{\xi\eta} + \gamma x_{\eta\eta} + x_{\xi}\Phi\alpha + x_{\eta}\Psi\gamma = 0$$
$$\alpha y_{\xi\xi} - 2\beta y_{\xi\eta} + \gamma y_{\eta\eta} + y_{\xi}\Phi\alpha + y_{\eta}\Psi\gamma = 0$$

Rearranging the equations above results in the final form of the Winslow equations using forcing functions in the Thomas-Middlecoff form:

$$\begin{aligned}\alpha\left(x_{\xi\xi} + \Phi x_{\xi}\right) - 2\beta x_{\xi\eta} + \gamma\left(x_{\eta\eta} + \Psi x_{\eta}\right) = 0\\ \alpha\left(y_{\xi\xi} + \Phi y_{\xi}\right) - 2\beta y_{\xi\eta} + \gamma\left(y_{\eta\eta} + \Psi y_{\eta}\right) = 0\end{aligned} \tag{B.16}$$

where again:

$$\alpha = x_{\eta}^{\,2} + y_{\eta}^{\,2}$$
$$\beta = x_{\xi}x_{\eta} + y_{\xi}y_{\eta}$$
$$\gamma = x_{\xi}^{\,2} + y_{\xi}^{\,2}$$

NOMENCLATURE

A	Area
$\vec{d}$	Direction vector about which to reflect a point
d_{AB}	Metric length
d_{lat}	Computational space offset in the lateral direction
d_n	Computational space offset in the normal direction
D	Drag
D*	Target drag
$\vec{e}_1$	First basis vector for transformed space (associated with normal spacing)
$\vec{e}_2$	Second basis vector for transformed space (associated with lateral spacing)
$\vec{F}$	Vector field
g	Geometric progression factor
h	Triangle height
I	Cost function
J	Grid Jacobian: $x_\xi y_\eta - x_\eta y_\xi$
J	Inverse Grid Jacobian: $\xi_x \eta_y - \xi_y \eta_x$
L	Lift
L*	Target lift
M	Riemannian metric tensor
$\vec{n}$	Outward facing normal
$\hat{n}$	Unit length normal vector
$\hat{n}_\xi$	ξ component of the unit normal vector in computational space
$\hat{n}_\eta$	η component of the unit normal vector in computational space
N	Valence count; i.e., the number of connected neighboring nodes
P_A	Average location of interior nodes connected to a boundary node
P_G	Ghost point
P_S	Surface point
P_{S-}	Surface point before P_S
P_{S+}	Surface point after P_S
$\hat{r}$	Edge vector used for refinement
R	Residual
T	Temperature

w_{11}	Coefficient for x component of local node 1 (the central node) on a triangle comprising a control volume
w_{21}	Coefficient for x component of local node 1 (the central node) on a triangle comprising a control volume
w_{12}	Coefficient for x component of local node 2 on a triangle comprising a control volume
w_{22}	Coefficient for x component of local node 2 on a triangle comprising a control volume
w_{13}	Coefficient for x component of local node 3 on a triangle comprising a control volume
w_{23}	Coefficient for x component of local node 3 on a triangle comprising a control volume
W_D	Drag coefficient
W_L	Lift coefficient
x_ξ	Derivative of the physical coordinate x with respect the computational coordinate ξ
x_η	Derivative of the physical coordinate x with respect the computational coordinate η
$x_{\xi\xi}$	Second derivative of x: $\partial^2 x / \partial \xi^2$
$x_{\eta\eta}$	Second derivative of x: $\partial^2 x / \partial \eta^2$
$x_{\xi\eta}$	Mixed second derivative of x: $\partial^2 x / \partial \eta \partial \xi$
y_ξ	Derivative of the physical coordinate y with respect the computational coordinate ξ
y_η	Derivative of the physical coordinate y with respect the computational coordinate η
$y_{\xi\xi}$	Second derivative of y: $\partial^2 y / \partial \xi^2$
$y_{\eta\eta}$	Second derivative of y: $\partial^2 y / \partial \eta^2$
$y_{\xi\eta}$	Mixed second derivative of y: $\partial^2 y / \partial \eta \partial \xi$
α	Winslow coefficient: $x_\eta^2 + y_\eta^2$
β	Winslow coefficient: $x_\xi x_\eta + y_\xi y_\eta$
β_i	Design variable
Δs	Off-body spacing

ϕ	Generic scalar, equipotential function
γ	Winslow coefficient: $x_\xi^2 + y_\xi^2$
Γ_i	Length of face i for a triangle
Λ	Scaling matrix (Eigenvalues of Riemannian metric tensor)
η	Second component of computational space
ω_z	Vorticity
ξ	First component of computational space
$\vec{\xi}_C$	Point specifically defined to be on a unit circle in computational space
ψ	Stream function